

PATENT ABSTRACTS OF JAPAN

(11)Publication number : 11-282701

(43)Date of publication of application : 15.10.1999

(51)Int.Cl.

G06F 11/10

G11B 20/18

H03M 13/00

(21)Application number : 10-324623

(71)Applicant : CIRRUS LOGIC INC

(22)Date of filing : 16.11.1998

(72)Inventor : CHRISTOPHER ZOCK

(30)Priority

Priority number : 97 970918

Priority date : 14.11.1997

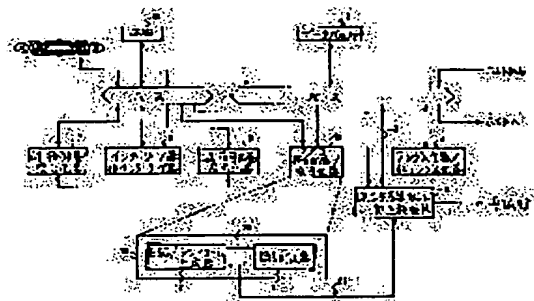
Priority country : US

(54) ECC SYSTEM FOR GENERATING CRC SYNDROME TO RANDOM DATA IN COMPUTER STORAGE DEVICE

(57)Abstract:

PROBLEM TO BE SOLVED: To avoid a waiting time that is related to access to a data buffer which is for nonrandomizing data before generating a cyclic redundancy check(CRC) syndrome by performing CRC check of randomized data.

SOLUTION: Randomized data is subjected to CRC check. In this error correction processor, an error checking and correction(ECC) decoder uses an ECC redundancy signal and corrects an error in the randomized data. A syndrome generator 17 responds to the randomized data and generates a verification syndrome. A correction verifier 19 compares the verification syndrome with prescribed value and verifies the effectiveness and completeness of correction to the randomized data. A nonrandomizer nonrandomizes the randomized data after the verifier 19 shows that correction of the randomized data is effective and complete.



LEGAL STATUS

[Date of request for examination]

[Date of sending the examiner's decision of rejection]

[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]

[Date of final disposal for application]

[Patent number]

Best Available Copy

1

【特許請求の範囲】

【請求項1】 ディスク記憶媒体から読み出されたランダム化されたデータ中の誤りを訂正するための誤り訂正プロセッサであって、該ランダム化されたデータは、該ランダム化されたデータに対して生成されたECC冗長記号と、ランダム化される前のデータに対して生成されたチェック記号とを含み、該誤り訂正プロセッサは、

(a) 該ECC冗長記号を用いて、該ランダム化されたデータ中の誤りを訂正するためのECC復号化器と、

(b) 該ランダム化されたデータに回答して、検証シンドロームを生成するためのシンドローム生成器と、

(c) 該検証シンドロームを所定の値と比較して、該ランダム化されたデータに対する訂正の有効性 (validity) および完全性を検証するための訂正検証器と、

(d) 該訂正検証器が、該ランダム化されたデータに対する訂正が有効で且つ完全であることを示した後、該ランダム化されたデータを非ランダム化するための非ランダム化器とを含む、誤り訂正プロセッサ。

【請求項2】 (a) 前記ランダム化されたデータが、交差する符号語の第1および第2の組を含み、

(b) 前記ECC復号化器が、該第1の組の符号語および該第2の組の符号語をシーケンシャルなパス (pass) で処理することによって、該ランダム化されたデータ中の誤りを訂正する、請求項1に記載の誤り訂正プロセッサ。

【請求項3】 (a) 前記シンドローム生成器が、前記第1の組の符号語に対する第1のパスの間にデータ検証シンドロームを生成し、

(b) 該シンドローム生成器が、前記ランダム化されたデータを訂正するために前記ECC復号化器によって生成される訂正值を用いて、誤り検証シンドロームを生成し、

(c) 該シンドローム生成器が、該データ検証シンドロームを該誤り検証シンドロームと結合して、前記所定の値と比較される最終検証シンドロームを生成する、請求項2に記載の誤り訂正プロセッサ。

【請求項4】 (a) 前記ECC復号化器が前記ランダム化されたデータを処理するのと同時に、前記シンドローム生成器が部分検証シンドロームを生成し、

(b) 該シンドローム生成器が、該ECC復号化器によって処理されている特定のランダム化されたデータ記号の位置に応じて、該部分検証シンドロームを調整するためのオフセットコントローラを含む、請求項2に記載の誤り訂正プロセッサ。

【請求項5】 前記ECC復号化器が、ランダム化されたデータ記号を訂正するための訂正值を生成し、

(a) 前記オフセットコントローラが、前記訂正されたデータ記号の位置に応じて前記部分検証シンドロームを調整し、

(b) 前記シンドローム生成器が、該訂正值を用いて該

2

部分検証シンドロームを更新する、請求項4に記載の誤り訂正プロセッサ。

【請求項6】 (a) 前記チェック記号が、有限領域の生成多項式 $G(x)$ に従って生成され、

(b) 前記オフセットコントローラが、前記部分検証シンドロームに $X^K \bmod G(x)$ を掛けることによって該部分検証シンドロームを調整し、ここで、 K はオフセット値であり、 \bmod 演算子は剰余除算 (modulo division) を行う、請求項4に記載の誤り訂正プロセッサ。

【請求項7】 (a) 前記チェック記号が、有限領域の生成多項式に従って生成され、

(b) 前記所定の値が、該生成多項式に基づく、請求項1に記載の誤り訂正プロセッサ。

【請求項8】 (a) 前記非ランダム化器が、所定のシード (seed) 値に従ってランダムパターンを生成し、

(b) 該ランダムパターンが、前記ランダム化されたデータと結合されて、該ランダム化されたデータを非ランダム化する、請求項1に記載の誤り訂正プロセッサ。

【請求項9】 (a) 前記チェック記号が、有限領域の生成多項式に従って生成され、

(b) 前記所定の値が、該生成多項式および前記ランダムパターンに基づく、請求項8に記載の誤り訂正プロセッサ。

【請求項10】 前記チェック記号が、巡回冗長符号 (CRC) に従って生成される、請求項1に記載の誤り訂正プロセッサ。

【請求項11】 ディスク記憶媒体から読み出されたランダム化されたデータ中の誤りを訂正する方法であって、該ランダム化されたデータは、該ランダム化されたデータに対して生成されたECC冗長記号と、ランダム化される前のデータに対して生成されたチェック記号とを含み、該方法は、

(a) 該ECC冗長記号を用いて、該ランダム化されたデータ中の誤りを訂正するステップと、

(b) 該ランダム化されたデータに回答して、検証シンドロームを生成するステップと、

(c) 該検証シンドロームを所定の値と比較して、該ランダム化されたデータに対する訂正の有効性および完全性を検証するステップと、

(d) 該訂正検証器が、該ランダム化されたデータに対する訂正が有効で且つ完全であることを示した後、該ランダム化されたデータを非ランダム化するステップとを包含する、方法。

【請求項12】 (a) 前記ランダム化されたデータが、交差する符号語の第1および第2の組を含み、

(b) 前記訂正するステップは、該第1の組の符号語および該第2の組の符号語をシーケンシャルなパスで処理することによって、該ランダム化されたデータ中の誤りを訂正する、請求項11に記載の方法。

【請求項13】 前記検証シンドロームを生成するステ

10

20

30

40

50

3

ップが、

(a) 前記第 1 の組の符号語に対する第 1 のパスの間にデータ検証シンドロームを生成するステップと、

(b) 前記ランダム化されたデータを訂正するための訂正值を用いて、誤り検証シンドロームを生成するステップと、

(c) 該データ検証シンドロームを該誤り検証シンドロームと結合して、前記所定の値と比較される最終検証シンドロームを生成するステップとを包含する、請求項 1 2 に記載の方法。

【請求項 1 4】 前記検証シンドロームを生成するステップが、

(a) 前記ランダム化されたデータを処理して E C C 誤りシンドロームを生成するのと同時に、部分検証シンドロームを生成するステップと、

(b) 該 E C C 誤りシンドロームを生成するために処理されている特定のランダム化されたデータ記号の位置に応じて、該部分検証シンドロームを調整するステップとを包含する、請求項 1 2 に記載の方法。

【請求項 1 5】 前記検証シンドロームを生成するステップが、

(a) 訂正されたデータ記号の位置に応じて前記部分検証シンドロームを調整するステップと、

(b) 前記データ記号を訂正するために用いられる前記訂正值で、該部分検証シンドロームを更新するステップとをさらに包含する、請求項 1 4 に記載の方法。

【請求項 1 6】 (a) 前記チェック記号が、有限領域の生成多項式 $G(x)$ に従って生成され、

(b) 前記部分検証シンドロームを調整する前記ステップが、該部分検証シンドロームに $X^K \bmod G(x)$ を掛けるステップを包含し、ここで、 K はオフセット値であり、 \bmod 演算子は剰余除算を行う、請求項 1 4 に記載の方法。

【請求項 1 7】 (a) 前記チェック記号が、有限領域の生成多項式に従って生成され、

(b) 前記所定の値が、該生成多項式に基づく、請求項 1 1 に記載の方法。

【請求項 1 8】 (a) 前記非ランダム化器が、所定のシード値に従ってランダムパターンを生成し、

(b) 該ランダムパターンが、前記ランダム化されたデータと結合されて、該ランダム化されたデータを非ランダム化する、請求項 1 1 に記載の方法。

【請求項 1 9】 (a) 前記チェック記号が、有限領域の生成多項式に従って生成され、

(b) 前記所定の値が、該生成多項式および前記ランダムパターンに基づく、請求項 1 8 に記載の方法。

【請求項 2 0】 前記チェック記号が、巡回冗長符号 (C R C) に従って生成される、請求項 1 1 に記載の方法。

【請求項 2 1】 ディスク記憶媒体から読み出されたデ

4

ータ中の誤りを訂正するための誤り訂正プロセッサであって、該データは、E C C 冗長記号とチェック記号とを含み、該誤り訂正プロセッサは、

(a) 該 E C C 冗長記号を用いて、該データ中の誤りを訂正するための E C C 復号化器と、

(b) 該データに回答して、検証シンドロームを生成するためのシンドローム生成器と、

(c) 該検証シンドロームを所定の値と比較して、該データに対する訂正の有効性および完全性を検証するための訂正検証器と、

(d) 該訂正検証器が、該ランダム化されたデータに対する訂正が有効で且つ完全であることを示した後、該データを非ランダム化するための非ランダム化器とを含む、誤り訂正プロセッサ。

【請求項 2 2】 ディスク記憶媒体から読み出されたランダム化されたデータ中の誤りを訂正するための誤り訂正プロセッサであって、該ランダム化されたデータは、該ランダム化されたデータに対して生成された E C C 冗長記号と、ランダム化される前のデータに対して生成されたチェック記号とを含み、該誤り訂正プロセッサは、

(a) 該ディスクから読み出された該ランダム化されたデータを格納するためのデータバッファと、

(b) 該データバッファから該ランダム化されたデータを受け取るように接続され、該ランダム化されたデータ中の誤りを訂正するために、E C C 誤りシンドロームおよび訂正值を生成するための E C C 復号化器と、

(c) 該 E C C 復号化器が該データバッファから該ランダム化されたデータを受け取るのと同時に該データバッファから該ランダム化されたデータを受け取るように接続され、検証シンドロームを生成するためのシンドローム生成器と、

(d) 該検証シンドロームを所定の値と比較して、該ランダム化されたデータに 2 3 に対する訂正の有効性および完全性を検証するための訂正検証器と、

(e) 該データバッファから該ランダム化されたデータを受け取るように接続され、該訂正検証器が、該ランダム化されたデータに対する訂正が有効で且つ完全であることを示した後、該ランダム化されたデータを非ランダム化するための非ランダム化器とを含む、誤り訂正プロセッサ。

【請求項 2 3】 前記検証シンドロームが、巡回冗長符号 (C R C) に従って生成される、請求項 2 2 に記載の誤り訂正プロセッサ。

【請求項 2 4】 前記シンドローム生成器が、ランダム化されたデータ記号が訂正されるときに前記 E C C 復号化器によって生成される訂正值を用いて、前記検証シンドロームを更新する、請求項 2 2 に記載の誤り訂正プロセッサ。

【発明の詳細な説明】

【0 0 0 1】

50

5

【発明の属する技術分野】本発明は、コンピュータ記憶装置のための誤り訂正システムに関し、特に、書き込み動作中にデータがランダム化される前に巡回冗長検査（CRC）シンボルが生成されるとき、ランダムデータに対してCRCを実施することによって、積符号などの多次元符号に対する訂正の有効性および完全性を証明するための効率的な方法および装置に関する。

【0002】

【従来の技術】コンピュータ記憶装置（磁気ディスクドライブおよび光ディスクドライブ）において、記録チャネルの帯域幅は、信号パワーと同様に限定される。性能利得を成し遂げるために、様々な符号化技術を用いてシステムのノイズに対する免疫を高めることによって効果的な信号対ノイズ比（SNR）を増加させている。このため、任意の低ビット誤り率を維持しながら記憶密度を増加させることによって、記憶容量を増加させることが可能となる。

【0003】現在、記録装置において用いられている符号には一般に2つのタイプがある。チャネル符号および誤り訂正符号（ECC）である。チャネル符号は、記録チャネルの特定の特徴によって引き起こされるノイズに対するものである。例えば、ランレングス限定（RL）符号は、アナログ搬送波信号におけるデータシンボルを示すパルス間の最小スペーシングを限定することによって、ノイズをシンボル間干渉に減衰するように設計されたチャネル符号である。記録されるデータのスペクトル内容は、読返し時にデータを正確に検出するシステムの能力にも悪影響を与え得る。この結果、いくつかのデータシーケンスは、他のデータシーケンスよりも検出が困難となり得る。この現象を考慮するために、通常、データをランダム化するためのチャネル符号が、記録装置において用いられている。このような記録装置は、データをディスク記憶媒体に書き込む前にランダム化することによってデータを効果的に「不透明にする（whiten）」する。読み返し時に、記録チャネルは、データがランダム化されない場合よりも低いビット誤り率でランダムデータを検出することができる。記憶媒体から読み出されたデータは、ホストに転送される前に非ランダム化される。

【0004】誤り訂正コーディング（ECC）において、記録されるバイナリデータは、数学的に処理され、データに追加される冗長シンボルを生成し、ディスク記憶媒体に書き込まれる符号語を形成する。読み返し時には、記録された符号ワードは、読出し信号から評価（検出）され、冗長シンボルは、評価された符号ワードを最初に記録されたユーザデータに復号化するために用いられる。実際、冗長シンボルは、符号ワードが記録チャネルを通過するときに、符号ワードをノイズから保護するバッファを提供する。十分なノイズがこのバッファに「浸透（penetrate）」すると、ノイズは、書き込まれ

6

た符号ワードを異なる受信符号ワードに変化させ、この結果、ユーザデータに復号化されるときに誤りを生じる。

【0005】誤り訂正符号においてより多くの冗長シンボルが用いられるほど、符号語の周囲のバッファは大きくなり、復号化誤りが発生する前に耐えられ得るノイズが多くなる。しかし、任意の低ビット誤り率を維持しながら所定のチャネルに対して達成可能な最大ユーザデータ転送率（または記録密度）を指す「チャネル容量」として公知の任意の所定の記録チャネルの性能には上限がある。最終的には、チャネル容量は、チャネル帯域幅および信号対ノイズ（SNR）比の関数である。上記のように、チャネル符号および誤り訂正符号は、効果的なSNRを増加させることによって性能を向上させる手段である。

【0006】記録チャネルの信頼性および効率を最大にするために、ユーザデータを符号化／復号化するための解決法が多数ある。最終目標は、実現を簡単にし、コストを下げると共に、チャネル容量に近づくシステムを設計することである。ブロック誤り訂正符号、特に、リード・ソロモンブロック符号は、その優れた誤り訂正特性および実現コストおよび複雑性が低いことにより、通常、ディスク記憶システムにおいて用いられる。

【0007】ブロック符号は、ソースデータストリームのkシンボル入力ブロックを、nシンボル出力ブロックまたは符号語に符号化する。ここで、n-kは、冗長シンボルの数である、k/nは、符号率と呼ぶ。次に、符号語は、通信媒体を通して転送され（通信媒体に記憶され）、受信器によって復号化される。符号化プロセスは、入力ブロックに対して数学演算を行い、出力符号語は、符号 d_{\min} の最小距離として参照されるパラメータによって他の符号語とは異なったものにされる。符号語間の最小距離 d_{\min} は、システムが、受信された符号語が誤って復号化される前の耐え得るノイズの量を決定する。

【0008】リード・ソロモン符号では、データストリームは、シンボルのシーケンスとして処理され、シンボルは、典型的には、有限フィールドGF(2^w)から選択される。パラメータwは、シンボル当たりのバイナリデータビットの数を示す。kシンボル入力ブロックの各シンボルは、データ多項式D(x)の係数を示す。次に、冗長シンボル（多項式W(x)とも呼ぶ）は、生成器多項式G(x)によって除算される入力データ多項式D(x)の剰余除算として計算される。

【0009】

【数1】

$$W(x) = (x^m \cdot D(x)) \text{ MOD } G(x)$$

ここで、mは、冗長シンボルの数と等しい生成器多項式の次数である。次に、冗長多項式W(x)は、データ多項式D(x)に加えられ、符号語多項式C(x)を生成

7

する。

【0010】

【数2】

$$C(x) = (x^m \cdot D(x)) + W(x).$$

当業者には言うまでもなく、上記の動作を行う符号化器回路は、リニアフィードバックシフトレジスタ (LFSR) を用いて最小のコストで実現され得る。

【0011】符号化の後、符号語 $C(x)$ は、ノイズ通信チャネルを通して転送され、受信された符号語 $C'(x)$ は、転送された符号語 $C(x)$ に誤り多項式 $E(x)$ を加えたものと等しくなる。受信された符号語 $C'(x)$ は、以下のステップに従って訂正される。

(1) 誤りシンドローム S_i を計算する。(2) 誤りシンドローム S_i を用いて誤りロケータ多項式の係数を計算する。(3) 誤りロケータ多項式のルートを計算する。ルートの対数は、誤り位置 L_i である。(4) 誤りシンドローム S_i および誤りロケータ多項式のルートを

用いて誤り値を計算する。

【0012】誤りシンドローム S_i は、生成器多項式 $G(x)$ の因数によって除算される受信符号語多項式 $C'(x)$ の剰余除算として計算される。

【0013】

【数3】

$$S_i = C'(x) \text{ MOD } (x + \alpha^i)$$

このとき

【0014】

【数4】

$$G(x) = \prod_{i=0}^{m-1} (x + \alpha^i)$$

であり、ここで、 α は、有限フィールド $GF(2^m)$ の基本要素である。復号化プロセスの他のステップを実施し、誤りロケータ多項式を計算し、誤りロケータ多項式のルートを計算し、誤り値を計算する技術は、当業者により周知であり、本発明を理解するために必要ではない。例えば、「COEFFICIENT UPDATING METHOD AND APPARATUS FOR REED-SOLOMON DECODER」という名称の下記で参照する米国特許第 5,446,743 号を参照のこと。

【0015】誤り許容度をさらに増加させるための当該技術分野で公知の他の技術としては、符号語を、多次元または積符号として知られるものに改作することが挙げられる。デジタルビデオディスク (DVD) 記憶システムは、例えば、一般に、図 3A に示す二次元積符号を用いる。符号語は、交差する水平 (行または Q) および垂直 (列または P) 符号語に改作され、復号化プロセスは、反復パスにおいて実行される。まず、水平符号語に対してパスが行われ、できるだけ多くの誤りが訂正される。訂正不可能な水平符号語はすべて、変更されずに残される。次に、垂直符号語に対してパスが行われ、でき

8

るだけ多くの誤りが訂正される。ここで、垂直符号語において訂正されるシンボルもまた、交差する水平符号語に対する対応のシンボルを訂正する。この結果、水平符号語は、次の水平パスにおいて訂正可能となり得る。同様に、水平パスにおいて訂正されるシンボルは、前回訂正不可能であった垂直符号語を、次の垂直パスにおいて訂正可能にし得る。この反復プロセスは、全積符号が訂正されるか、または訂正不可能になるまで続行される。

【0016】図 3A の二次元積符号はさらに、行および列符号語に対する訂正の有効性を検査するのに用いられる CRC 冗長シンボルを有する。CRC 冗長は、典型的には、以下の式に従ってユーザデータを処理することによって生成される。

【0017】

【数5】

$$CRC \text{ 冗長} = P(x) \cdot x^{n-k} \text{ mod } G(x)$$

ここで、 $P(x)$ は、有限フィールド $GF(2^m)$ における係数を有する多項式として示されるユーザデータであり、 $n-k$ は、CRC 冗長シンボルの数であり、 $G(x)$ は、生成器多項式である。次に、CRC 冗長は、得られる符号語 $C(x)$ がディスクに書き込まれる前にユーザデータに追加される。読出し動作中、ディスクから読み出されたデータは、以下の式に従って処理され、CRC シンドローム S_{CRC} を生成する。

【0018】

【数6】

$$S_{CRC} = C'(x) \text{ mod } G(x).$$

ここで、 $C'(x)$ は、ディスクから読み出された (CRC 冗長を含む) 受信符号語多項式である。符号語 $C'(x)$ に誤りがない場合、シンドローム S_{CRC} はゼロになる。

【0019】CRC 冗長は、典型的には、ECC 冗長シンボルを符号化する前に書込み動作中にデータに対して生成され、CRC シンドロームは、ECC 冗長が積符号を訂正するために用いられた後、読出し動作中に生成される。このように、CRC シンドロームは、訂正を検証し、訂正ミスを検出するように動作する。これは、非常に重要な機能である。なぜなら、この機能によって、誤り訂正システムが、「悪いデータ」がホストシステムを通過することを防止するためである。

【0020】図 1 は、CD/DVD 光ディスク記憶システムにおいて典型的に見いだされる従来の誤り訂正システムの概観を示す。書込み動作中 (装置は読出し専用でないと仮定する)、ホストシステムから受信されたユーザデータは、データバッファ 1 に記憶される。次に、CRC 生成器および訂正検証器 2 は、ユーザデータをバッファからライン 3 を介して読み出し、CRC 冗長シンボルを生成し、冗長シンボルが追加されたユーザデータをデータバッファ 1 に再び記憶する。その後、データ (CRC 冗長を含む) は、データバッファ 1 から再び読み出

9

され、データランダム化器 4 によってランダム化にされ、ランダムデータは、データバッファ 1 に再び記憶される。次に、P/Q 符号化器/復号化器 5 は、データバッファ 1 からランダムデータを読み出し、ECC/シンドローム生成器 12 は、P および Q 符号語に対して ECC 冗長シンボルを生成し、図 3 A に示す二次元積符号を形成する。個別の P および Q 符号語は、ECC 冗長シンボルを追加した後、データバッファ 1 に再び記憶される。全積符号が生成されると、全積符号は、データバッファ 1 から読み出され、光記憶媒体 6 に書き込まれる。

【0021】システムが、コンパクトディスク (CD) データフォーマット用に構成される場合、C1 および C2 として参照されるさらなる冗長が生成され、データがディスクに書き込まれる前に、データに追加される。従って、CD 記録フォーマットを容易にするために、誤り訂正システムは、C1 符号化器/復号化器 7、C2 符号化器/復号化器 8、および周知のクロスインターリーブリード-ソロモン符号 (CIRC) を実現するためのインターリーブ器/非インターリーブ器 9 を有する。典型的には、スタティック RAM (SRAM) 10 は、CIRC コーディングプロセスを実現するために用いられ、SRAM は、ダイナミック RAM (DRAM) よりもはるかに速く、後者は、データバッファ 1 を実現するために用いられる。

【0022】読出し動作中、プロセスは逆に進行する。CD フォーマット用に構成する場合、C1 および C2 復号化器は、ランダムデータが光ディスク 6 から読み出され、データバッファ 1 に記憶されるときに、ランダムデータに予め訂正を行う。完全な積符号が、データバッファ 1 で利用できるようになると、P/Q 復号化器 5 は、P および Q 符号語に対して反復パスを開始し、さらなる訂正を行う。ECC/シンドローム生成器は、ライン 13 を介して誤り訂正器 14 に転送される ECC シンドロームを生成する。誤り訂正器は、ECC シンドロームを用いて、上記のように、個々の符号語における誤りを訂正する。P または Q パスの終わりには、ECC 誤りシンドロームはすべてゼロになり、これは、積符号が、(訂正ミスがなければ) 誤りを含まないことを意味し、ランダムデータは、データバッファ 1 から読み出され、非ランダム化器 4 によって非ランダム化される。データが非ランダム化される間、CRC 生成器および訂正検証器 2 によって処理され、CRC シンドロームを生成する。CRC シンドロームがゼロである場合、これは、P および Q 符号語に対する訂正は有効かつ完了していることを意味し、データは、再び、データバッファ 1 から読み出され、非ランダム化され、非ランダム化されたデータは、ホストシステムに転送される。CRC シンドロームがゼロでない場合、これは、積符号の訂正ミスがあったことを示し、CRC 生成器および訂正検証器 2 は、誤りメッセージをライン 11 を介してホストシステムに転送し、

10

ホストシステムは、再試行動作 (即ち、ディスクからデータを再び読み出す試み) を開始する。

【0023】

【発明が解決しようとする課題】図 1 に示す従来の誤り訂正システムの基本的な欠点は、データが、CRC 検証ステップを実施する前に、非ランダム化されなければならないことである。このためには、上記のように全積符号を読み出し、データを非ランダム化し、CRC シンドロームを生成するためにさらにバッファにアクセスしなければならない。明らかに、これは、記憶システムの待ち時間を増加させ、特に、円滑で中断されない性能を成し遂げるために、音声/映像データの大きなブロックが、連続したストリームにおいて記憶システムから読み出されなければならないマルチメディアアプリケーションにはあまり望ましくない。

【0024】従って、検証シンボルがランダム化の前に生成され、ECC シンボルがランダム化の後に生成されるとき、CD/DVD 積符号などの多次元符号に対する訂正の有効性および完全性の証明に関連する待ち時間を避けるコンピュータ記憶装置における誤り訂正システムが求められている。

【0025】

【課題を解決するための手段】本発明による誤り訂正プロセッサは、ディスク記憶媒体から読み出されたランダム化されたデータ中の誤りを訂正するための誤り訂正プロセッサであって、該ランダム化されたデータは、該ランダム化されたデータに対して生成された ECC 冗長記号と、ランダム化される前のデータに対して生成されたチェック記号とを含み、該誤り訂正プロセッサは、

(a) 該 ECC 冗長記号を用いて、該ランダム化されたデータ中の誤りを訂正するための ECC 復号化器と、
(b) 該ランダム化されたデータに回答して、検証シンドロームを生成するためのシンドローム生成器と、
(c) 該検証シンドロームを所定の値と比較して、該ランダム化されたデータに対する訂正の有効性 (validity) および完全性を検証するための訂正検証器と、
(d) 該訂正検証器が、該ランダム化されたデータに対する訂正が有効で且つ完全であることを示した後、該ランダム化されたデータを非ランダム化するための非ランダム化器とを含んでおり、これにより上記目的が達成される。

【0026】(a) 前記ランダム化されたデータが、交差する符号語の第 1 および第 2 の組を含み、(b) 前記 ECC 復号化器が、該第 1 の組の符号語および該第 2 の組の符号語をシーケンシャルなパス (pass) で処理することによって、該ランダム化されたデータ中の誤りを訂正してもよい。

【0027】(a) 前記シンドローム生成器が、前記第 1 の組の符号語に対する第 1 のパスの間にデータ検証シンドロームを生成し、(b) 該シンドローム生成器が、

11

前記ランダム化されたデータを訂正するために前記 ECC 復号化器によって生成される訂正值を用いて、誤り検証シンδροームを生成し、(c) 該シンδροーム生成器が、該データ検証シンδροームを該誤り検証シンδροームと結合して、前記所定の値と比較される最終検証シンδροームを生成してもよい。

【0028】(a) 前記 ECC 復号化器が前記ランダム化されたデータを処理するのと同時に、前記シンδροーム生成器が部分検証シンδροームを生成し、(b) 該シンδροーム生成器が、該 ECC 復号化器によって処理されている特定のランダム化されたデータ記号の位置に応じて、該部分検証シンδροームを調整するためのオフセットコントローラを含んでもよい。

【0029】前記 ECC 復号化器が、ランダム化されたデータ記号を訂正するための訂正值を生成し、(a) 前記オフセットコントローラが、前記訂正されたデータ記号の位置に応じて前記部分検証シンδροームを調整し、

(b) 前記シンδροーム生成器が、該訂正值を用いて該部分検証シンδροームを更新してもよい。

【0030】(a) 前記チェック記号が、有限領域の生成多項式 $G(x)$ に従って生成され、(b) 前記オフセットコントローラが、前記部分検証シンδροームに $X^K \bmod G(x)$ を掛けることによって該部分検証シンδροームを調整し、ここで、 K はオフセット値であり、 \bmod 演算子は剰余除算 (modulo division) を行ってもよい。

【0031】(a) 前記チェック記号が、有限領域の生成多項式に従って生成され、(b) 前記所定の値が、該生成多項式に基づいてもよい。

【0032】(a) 前記非ランダム化器が、所定のシード (seed) 値に従ってランダムパターンを生成し、

(b) 該ランダムパターンが、前記ランダム化されたデータと結合されて、該ランダム化されたデータを非ランダム化してもよい。

【0033】(a) 前記チェック記号が、有限領域の生成多項式に従って生成され、(b) 前記所定の値が、該生成多項式および前記ランダムパターンに基づいてもよい。

【0034】前記チェック記号が、巡回冗長符号 (CRC) に従って生成されてもよい。

【0035】本発明による方法は、ディスク記憶媒体から読み出されたランダム化されたデータ中の誤りを訂正する方法であって、該ランダム化されたデータは、該ランダム化されたデータに対して生成された ECC 冗長記号と、ランダム化される前のデータに対して生成されたチェック記号とを含み、該方法は、(a) 該 ECC 冗長記号を用いて、該ランダム化されたデータ中の誤りを訂正するステップと、(b) 該ランダム化されたデータに

(c) 該検証シンδροームを所定の値と比較して、該ラ

12

ンダム化されたデータに対する訂正の有効性および完全性を検証するステップと、(d) 該訂正検証器が、該ランダム化されたデータに対する訂正が有効かつ完全であることを示した後に、該ランダム化されたデータを非ランダム化するステップとを包含しており、これにより、上記目的が達成される。

【0036】(a) 前記ランダム化されたデータが、交差する符号語の第 1 および第 2 の組を含み、(b) 前記訂正するステップは、該第 1 の組の符号語および該第 2 の組の符号語をシーケンシャルなパスで処理することによって、該ランダム化されたデータ中の誤りを訂正してもよい。

【0037】前記検証シンδροームを生成するステップが、(a) 前記第 1 の組の符号語に対する第 1 のパスの間にデータ検証シンδροームを生成するステップと、

(b) 前記ランダム化されたデータを訂正するための訂正值を用いて、誤り検証シンδροームを生成するステップと、(c) 該データ検証シンδροームを該誤り検証シンδροームと結合して、前記所定の値と比較される最終検証シンδροームを生成するステップとを包含してもよい。

【0038】前記検証シンδροームを生成するステップが、(a) 前記ランダム化されたデータを処理して ECC 誤りシンδροームを生成するのと同時に、部分検証シンδροームを生成するステップと、(b) 該 ECC 誤りシンδροームを生成するために処理されている特定のランダム化されたデータ記号の位置に応じて、該部分検証シンδροームを調整するステップとを包含してもよい。

【0039】前記検証シンδροームを生成するステップが、(a) 訂正されたデータ記号の位置に応じて前記部分検証シンδροームを調整するステップと、(b) 前記データ記号を訂正するために用いられる前記訂正值で、該部分検証シンδροームを更新するステップとをさらに包含してもよい。

【0040】(a) 前記チェック記号が、有限領域の生成多項式 $G(x)$ に従って生成され、(b) 前記部分検証シンδροームを調整する前記ステップが、該部分検証シンδροームに $X^K \bmod G(x)$ を掛けるステップを包含し、ここで、 K はオフセット値であり、 \bmod 演算子は剰余除算を行ってもよい。

【0041】(a) 前記チェック記号が、有限領域の生成多項式に従って生成され、(b) 前記所定の値が、該生成多項式に基づいてもよい。

【0042】(a) 前記非ランダム化器が、所定のシード値に従ってランダムパターンを生成し、(b) 該ランダムパターンが、前記ランダム化されたデータと結合されて、該ランダム化されたデータを非ランダム化してもよい。

【0043】(a) 前記チェック記号が、有限領域の生成多項式に従って生成され、(b) 前記所定の値が、該

13

生成多項式および前記ランダムパターンに基づいてもよい。

【0044】前記チェック記号が、巡回冗長符号（CRC）に従って生成されてもよい。

【0045】本発明による別の誤り訂正プロセッサは、ディスク記憶媒体から読み出されたデータ中の誤りを訂正するための誤り訂正プロセッサであって、該データは、ECC冗長記号とチェック記号とを含み、該誤り訂正プロセッサは、（a）該ECC冗長記号を用いて、該データ中の誤りを訂正するためのECC復号化器と、

（b）該データに应答して、検証シンドロームを生成するためのシンドローム生成器と、（c）該検証シンドロームを所定の値と比較して、該データに対する訂正の有効性および完全性を検証するための訂正検証器と、

（d）該訂正検証器が、該ランダム化されたデータに対する訂正が有効で且つ完全であることを示した後、該データを非ランダム化するための非ランダム化器とを含んでおり、これにより上記目的が達成される。

【0046】本発明による別の誤り訂正プロセッサは、ディスク記憶媒体から読み出されたランダム化されたデータ中の誤りを訂正するための誤り訂正プロセッサであって、該ランダム化されたデータは、該ランダム化されたデータに対して生成されたECC冗長記号と、ランダム化される前のデータに対して生成されたチェック記号とを含み、該誤り訂正プロセッサは、（a）該ディスクから読み出された該ランダム化されたデータを格納するためのデータバッファと、（b）該データバッファから該ランダム化されたデータを受け取るように接続され、該ランダム化されたデータ中の誤りを訂正するために、ECC誤りシンドロームおよび訂正值を生成するためのECC復号化器と、（c）該ECC復号化器が該データバッファから該ランダム化されたデータを受け取るのと同時に該データバッファから該ランダム化されたデータを受け取るように接続され、検証シンドロームを生成するためのシンドローム生成器と、（d）該検証シンドロームを所定の値と比較して、該ランダム化されたデータに対する訂正の有効性および完全性を検証するための訂正検証器と、（e）該データバッファから該ランダム化されたデータを受け取るように接続され、該訂正検証器が、該ランダム化されたデータに対する訂正が有効で且つ完全であることを示した後、該ランダム化されたデータを非ランダム化するための非ランダム化器とを含んでおり、これにより上記目的が達成される。

【0047】前記検証シンドロームが、巡回冗長符号（CRC）に従って生成されてもよい。

【0048】前記シンドローム生成器が、ランダム化されたデータ記号が訂正されるときに前記ECC復号化器によって生成される訂正值を用いて、前記検証シンドロームを更新してもよい。

【0049】多次元符号に対する訂正の有効性および完

14

全性の証明に関連する待ち時間を避けるコンピュータ記憶装置のための誤り訂正システムが開示される。好ましい実施態様において、検証は、巡回冗長検査（CRC）を用いて実施される。書き込み動作中、CRC冗長シンボルは、ホストシステムから受信したユーザデータに対して計算され、CRCシンボルを追加した後、データは、疑似ランダムデータパターンで排他的論理和計算することによってランダム化される。次に、ECCシンボルは、ランダム化データに対して（好ましくは、リードノンロモン符号を用いて）生成され、行（Q）および列

（P）符号語の積符号を形成する。次に、この積符号は、ディスクに書き込まれる。読み返し時、積符号は、データバッファに記憶され、P/Q復号化器によって復号化される。Q符号語に対する第1パスの間、データCRCシンドロームは、訂正されていないランダム化データに対して生成され、データCRCシンドロームは、データCRCレジスタに記憶される。さらに、第1パスおよび次のパスにおいては、PまたはQ符号語に訂正がなされるとき、訂正值は、誤りCRCレジスタに適用される。完全なCRC符号語を処理した後、データCRCレジスタと誤りCRCレジスタとは組み合わせられ、最終CRCシンドロームが生成される。最終CRCシンドロームは、定数と比較され、積符号に対する訂正が有効かつ完了しているかどうかを決定する。このとき、定数は、ランダムデータパターンに対するCRCと等しい。このように、CRC検査は、ランダム化データに対して行われ得る。これによって、CRCシンドロームを生成する前にデータを非ランダム化するためのデータバッファへのアクセスに関連する待ち時間が避けられる。

【0050】本発明によって提供される他の利点は、PおよびQ符号語の訂正と同時に実行中にCRCシンドロームを生成する能力である。従って、CRC符号語を処理した直後に、CRCシンドロームは、訂正の有効性および完全性を検査するために利用可能である。つまり、CRCシンドロームを生成するためにデータバッファにアクセスする必要はないのである。

【0051】本発明の実施可能性局面は、PおよびQパス中にデータおよび誤りCRCシンドロームを調節し、CRC符号語シンボル内のオフセットを補うことである。例えば、垂直（即ち、P）符号語を処理する場合、処理される各垂直シンボルに対して、1行のデータシンボルによって誤りCRCシンドロームを調節する必要がある。これは、以下の式によってデータおよび誤りCRCシンドロームを乗算する特殊な乗算器回路を用いて実行される。

【0052】

【数7】

$$x^k \text{ MOD } G(x)$$

ここで、kは、オフセット（例えば、1行のシンボル）、およびG(x)は、CRC生成器多項式である。

10

20

30

40

50

15

【0053】各データセクタに追加された個別のCRCシンボルを有する多数のデータセクタを備えたDVD積符号については、CDモードにおいてC1およびC2符号化／復号化に用いられるSRAMが、各データセクタに対する部分データおよび誤りCRCシンドロームを記憶するために用いられる。PおよびQパスの間、データおよび誤りCRCシンドロームレジスタは、P/Q復号化器によって処理される現在のデータシンボルに応じて、適切な部分CRCシンドロームでロードされる。各データセクタを処理した後、各データセクタに対するデータCRCシンドロームと誤りCRCシンドロームとは組み合わせられ、疑似ランダムデータパターンに対するCRCと等しい定数と比較される。

【0054】本発明の上記ならびに他の局面および利点は、以下の本発明の詳細な説明を図面を参照しながら読むことによってより明確に理解される。

【0055】

【発明の実施の形態】なお本発明の優先権主張の基礎となる米国出願は、「AN ECC SYSTEM EMPLOYING A DATA BUFFER FOR STORING CODEWORD DATA AND A SYNDROME BUFFER FOR STORING ERROR SYNDROMES」という名称の同時に出願された米国特許出願第08/970,730号、ならびに「CONCURRENT GENERATION OF ECC ERROR SYNDROMES AND CRC VALIDATION SYNDROMES IN A DVD STORAGE DEVICE」という名称の同時に出願された米国特許出願第08/970,600号および「COEFFICIENT UPDATING METHOD AND APPARATUS FOR REED-SOLOMON DECODER」という名称の米国特許第5,446,743号である。上記の米国特許出願および米国特許は、本願では、参考のために援用する。

【0056】【システムの概観】図2は、本発明の誤り訂正システムの概観を示す。このシステムの動作は、以下の変更を除いて、図1を参照しながら上述した従来のシステムと同様である。読出し動作中、ECC誤りシンドロームは、第1水平パスにおいて、積符号の水平符号語および垂直符号語の両方に対して同時に生成される。ECC誤りシンドロームは、SRAM15内に記憶され、SRAM15はCD積符号を復号化するときにはCIRC誤り訂正用にも用いられる。これにより、記憶装置の待ち時間が大幅に減少する。なぜなら、次の水平または垂直パスの間にECC誤りシンドロームを再生するためにデータバッファ1にアクセスする必要がないからである。本発明の他の重要な変更は、積符号の訂正と同時にCRCシンドロームを生成すること、およびデータバッファに記憶されたデータを非ランダム化する前にCRCシンドロームを検査することである。これにより、CRCシンドロームを生成するために、データバッファ1から積符号全体を読み出す必要がなくなり、記憶システムの待ち時間はさらに減少する。

【0057】図1の従来の誤り訂正システムと異なる構

16

成要素は、SRAM15、P/Q復号化器16(ECC/シンドローム生成器17および誤り訂正器18を含む)、およびCRC生成器および訂正検証器19である。ECC/シンドローム生成器17は、ライン20を介して誤り訂正器に転送されるECCシンドロームを生成および記憶する際にSRAM15を用いる。データバッファ1に記憶される符号語を訂正するための誤り訂正器18によって生成される訂正值は、ECCシンドロームを更新する際に使用されるECC/シンドローム生成器17にもライン21を介して通信される。CRC生成器および訂正検証器19はまた、SRAM15を用いて、DVD積符号の16個のデータセクタに対する16個の部分CRCシンドロームを記憶する。SRAM15内に記憶される部分CRCシンドロームは、誤り訂正器18によって生成される訂正值を用いて更新される。訂正值は、ライン21を介して、CRC生成器および訂正検証器19と通信される。

【0058】【データフォーマット】図3Aは、DVD記憶装置において典型的に用いられる2次元積符号のデータフォーマットを示す。積符号は、16個のデータセクタを有し、各データセクタは、12個の水平符号語(Q符号語)を有する。各水平符号語は、好ましくはリード-ソロモン符号に従って生成される10個のECC冗長シンボルを有する。182個の垂直符号語(P符号語)があり、それぞれ、図示するように、16個の冗長シンボルを有する。ECC冗長シンボルは、ECC符号語も形成する。即ち、ECC冗長シンボルは、ユーザデータと同様に訂正可能である。従って、右側には、10個のECC符号語を含む全部で182個の垂直符号語があり、下側には、16個の水平ECC符号語を含む208個の水平符号語がある。

【0059】16個のデータセクタのそれぞれの端部には、ECC冗長シンボルを使用して符号語に対する訂正の有効性および完全性を証明するのに用いられる4個のCRCシンボルがある。上記のように、書込み動作中、CRCシンボルは、通常、データがランダム化される前およびECC冗長シンボルを加える前に、ユーザデータに対して生成される。従って、CRCシンボルは、ECCシンボルをカバーしない。さらに、従来の誤り訂正システムは、CRC検査を行う前にデータを非ランダム化しなければならない。

【0060】図3Bは、図3Aの積符号の最初の2個のデータセクタのさらなる詳細を示す。ECC符号語の訂正と同時にCRCシンドロームを生成する本発明の技術について、図3Bを参照しながら以下に説明する。

【0061】【データランダム化器/非ランダム化器】本発明の誤り訂正システムは、ユーザデータを不透明にし、検出するのが困難なデータシーケンスがディスクに記録されるのを防止する。図2を参照しながら上述したように、データバッファ1に記憶されるユーザデータ

17

は、CRCシンボルが生成された後にランダム化される。次に、ECC/シンドローム生成器 12 は、ランダム化データを処理し、図 3 A に示す積符号の P および Q 符号語に対する ECC 冗長を生成する。読み返し時に、積符号は訂正され、訂正が確認される。訂正が有効かつ完了している場合、データは非ランダム化され、ホストシステムに転送される。

【0062】データをランダム化/非ランダム化するための回路は当業者に周知であり、図 3 C および図 3 D は、この回路を示す。回路は、ライン 23 に与えられる疑似ランダムデータシーケンスを生成するためのランダムパターン生成器を有する。読出し動作中、データバッファ 1 に記憶されるユーザデータおよび CRC シンボルは、加算器 24 においてランダムデータシーケンスに加えられ（排他的論理和計算）、それによって、データがディスクに書き込まれる前にデータをランダム化する。読み返し時に、ランダムパターン生成器 22 は、加算器 25 において、ディスクから読み出されたデータに加えられ（排他的論理和計算）、それによって、データをホストシステムに転送する前にデータを非ランダム化する。

【0063】好ましくは、疑似ランダムデータシーケンスは、8 ビットシンボル（ECC 符号のシンボルサイズ）を用いて生成される。疑似ランダムデータシーケンスを生成するための好ましい実施態様は、図 3 D に示すように、リニアフィードバックシフトレジスタ（LFSR）を用いることである。LFSR 回路は、ライン 26 を介してシード値で初期化され、疑似ランダムデータシーケンスは、シード値に応じて異なる。異なるシード値は、図 3 A に示す各 DVD 積符号に対して用いられる。

【0064】[水平および垂直 ECC シンドロームの同時生成] ECC および CRC シンドロームを同時に生成し、訂正値を用いて ECC および CRC シンドロームを更新する回路および流れ図を以下に開示する。本発明の ECC シンドローム生成器 17 について 2 つの実施態様がある。第 1 の実施態様では、水平バスの間に水平符号語に対してシンドロームが生成されるのと同時に、ECC シンドロームが、垂直符号バスに対して生成される。以下にさらに詳細に記載するように、SRAM 15 は、垂直シンドロームの生成を容易にする。このように、垂直符号語に対する ECC シンドロームが、水平バスの直後に SRAM 15 内において利用できるので、垂直符号語は、ECC シンドロームを生成するためにデータバッファにアクセスせずに訂正され、それによって、誤り訂正待ち時間が大幅に減少する。本実施態様は、訂正が、通常、水平および垂直符号語に対する一回のバスの後完了する、DVD 記憶装置などにおけるように、大量の ECC 冗長を用いる積符号に特に適している。

【0065】ECC シンドローム生成器 17 の第 2 の実施態様は、より少ない ECC 冗長を用いる積符号に向け

18

られており、そのため、多数の水平および垂直バスを必要とする。この実施態様では、SRAM 15 は、水平および垂直符号語の両方に対して ECC シンドロームを記憶する。ECC シンドロームの両セットは、第 1 水平バスの間に同時に生成され、ECC シンドロームは、訂正値を用いて更新される。このように、データバッファにアクセスし、ECC シンドロームを再生成する際の待ち時間は、水平および垂直バスの両方に対して避けられる。誤り訂正は、わずかな時間で行われる。なぜなら、第 1 バスに続くバスは、データを訂正するためにデータバッファにアクセスすることのみが必要であるからである。

【0066】本発明の第 1 の実施態様では、垂直誤りシンドロームのみが SRAM 15 内に記憶されるが、これは、図 4 および図 5 を参照することによって理解される。図 4 は、各水平バスにおける水平符号語に対する誤りシンドロームを生成するために用いられる回路を示す。即ち、水平誤りシンドロームは常に再生成され、SRAM 15 内には記憶されない。水平誤りシンドローム S_i を生成するために、図 4 の回路は、以下の生成器多項式 $G(x)$ の因数によって各水平符号語 $C'(x)$ の剰余除算を計算する。

【0067】

【数 8】

$$S_i = C'(x) \text{ MOD } (x + \alpha^i)$$

このとき

【0068】

【数 9】

$$G(x) = \prod_{i=0}^{m-1} (x + \alpha^i)$$

である。この計算を行うために、水平符号語（ECC 冗長を含む）のシンボルは、データバッファ 1 から連続して読み出され、ライン 27 を介して、リニアフィードバックシフトレジスタ（LFSR）280 から 289 のバンクに与えられる。好ましい実施態様において、各水平符号語は、図 3 A に示すように、10 個の ECC 冗長シンボルを含むので、図 4 では 10 個の LFSR が存在することになる。各 LFSR には、フィードバックバスにおいて、対応する α^i 係数乗算器が設けられている。各 LFSR は、生成器多項式 $G(x)$ の各因数について剰余除算を行い、これによって、上式についての誤りシンドローム S_i を生成する。図 4 に開示する回路は、当業者に周知であり、本発明の新規の局面は、垂直符号語に対して誤りシンドロームを同時に生成することにある。図 5 は、この詳細を示す。

【0069】数学的に、垂直符号語に対する誤りシンドロームは、上記の水平符号語に対するのと同様に計算される。即ち、垂直誤りシンドローム S_i は、生成器多項式 $G(x)$ の因数で各垂直符号語 $C'(x)$ の剰余除算

19

を計算することによって生成される。従来のシンドローム生成器は、通常、図 4 に示すのと同じ回路を用いて垂直誤りシンドロームを生成する。即ち、垂直符号語（ECC 冗長を含む）のシンボルは、データバッファ 1 から連続して読み出され、LFSR のバンクを通してシフトされる。本発明において、垂直誤りシンドロームは、水平誤りシンドロームの生成と同時に生成され、垂直バスの間に垂直符号語を読み出すためにデータバッファへアクセスすることが避けられる。

【0070】図 5 は、垂直誤りシンドロームを同時に生成するための回路を示す。この回路の動作は、図 3 A に示す積符号を参照することによって理解される。SRAM 15 は、182 個の垂直符号語のそれぞれに対して 16 個の誤りシンドローム S_i を記憶する容量を有する。SRAM 15 内における垂直誤りシンドローム S_i は、第 1 水平バスの初めにゼロに初期化される。第 1 水平符号語を処理する際、シンボルは、データバッファ 1 から連続して読み出され、ライン 27 を介して図 4 の LFSR に与えられ、水平誤りシンドロームを生成する。シンボルは、同時に、ライン 27 を介して図 5 に示す回路に与えられ、垂直誤りシンドロームを生成する。図 4 と同様に、図 5 は、生成器多項式 $G(x)$ の 16 個の因数によって垂直符号語シンボル（各垂直符号語は、16 個 ECC 冗長シンボルを有する）の剰余除算を計算するためのシンドローム生成回路 290 から 2915 のバンクを有する。

【0071】図 5 に示す個々のシンドローム生成回路 290 から 2915 を理解するために、第 1 水平符号語がデータバッファから読み出されるとき動作について考えよう。第 1 水平符号語の第 1 シンボルは、第 1 垂直符号語の第 1 シンボルに対応する。従って、制御ライン 30 は、第 1 垂直符号語に対して、16 個の垂直誤りシンドローム（各 8 ビット）を SRAM 15 から取り出す。各 8 ビット垂直 ECC シンドロームは、対応するレジスタ 310 から 3115 にラッチされ、乗算器 320 から 3215 の出力として選択され、対応する α^i フィードバック係数 330 から 3315 によって乗算される。ライン 27 上の符号語シンボルは、乗算器 340 から 3415 の出力として選択され、加算器 350 から 3515 において、係数乗算器の出力として加算される。加算器 350 から 3515 の出力で更新されたシンドロームは、次に、SRAM 15 内に再記憶される。第 1 水平符号語の第 2 シンボルを処理する際、制御ライン 30 は、第 2 の垂直符号語に対して、16 個垂直誤りシンドロームを SRAM 15 から取り出し、上記の手順が繰り返される。このプロセスは、水平符号語のそれぞれに対して続行し、水平バスの終わりには、垂直符号語を訂正するための誤りシンドロームが、SRAM 15 内に記憶される。

【0072】水平バスの間に水平符号語が訂正される場合、SRAM 15 内に記憶されている対応する垂直誤り

20

シンドロームは、訂正されたシンボルを考慮するために更新されなければならない。図 2 のシンドローム生成器 17 および誤り訂正器 18 は、好ましくは、隣接する符号語上で動作する。換言すると、シンドローム生成器 17 が、現在の水平符号語に対して誤りシンドロームを生成している間、誤り訂正器 18 は、前の水平符号語に対してシンボルを訂正する。さらに、誤り訂正は、水平誤りシンドロームの生成を追跡し、訂正值で垂直誤りシンドロームの調整を単純にする。

【0073】例えば、シンドローム生成器 17 が、誤り訂正器 18 が、第 1 の水平符号語を訂正する間、図 3 A の第 2 水平符号語に対して誤りシンドロームを生成することについて考慮する。シンドローム生成器 17 が、第 2 符号語の第 3 シンボル 36 を通過し、第 1 符号語の第 3 シンボル 37 に誤りがあると仮定する。誤り訂正器 18 は、データバッファ 1 に記憶されている第 1 水平符号語の第 3 シンボル 37 を訂正するために用いられる訂正值を生成し、訂正值は、図 5 の垂直シンドローム生成回路 290 から 2915 にライン 21 を介して与えられる。

制御ライン 30 は、レジスタ 310 から 3115 にラッチされる、第 3 垂直符号語に対する垂直誤りシンドロームを SRAM 15 から取り出す。次に、レジスタ 310 から 3115 の出力は、加算器 350 から 3515 に対する入力として、乗算器 340 から 3415 を通して選択される。ライン 21 を介して与えられる訂正值は、乗算器 320 から 3215 の出力として選択され、対応の α^i フィードバック係数 330 から 3315 によって乗算され、加算器 350 から 3515 において垂直誤りシンドロームに加算される。訂正值を対応する α^i フィードバック係数 330 から 3315 によって乗算することは、現在の垂直 ECC シンドローム値と、訂正されているシンボルとの間のオフセット（即ち、垂直符号語における 1 つのシンボルのオフセット）を考慮するのに必要である。

【0074】水平バスの終わりには、垂直符号語に対する誤りシンドロームは、完全に生成され、即座に処理に利用できる。従って、垂直バスを実行するためには、垂直誤りシンドロームは、SRAM 15 から単に取り出され、垂直符号語を訂正するために誤り訂正器 18 によって使用される。垂直バスの後、積符号の CRC シンボルが、誤りがまだ残っていることを示す場合、上記のプロセスが繰り返される（即ち、水平および垂直誤りシンドロームが、次の水平バスにおいて再生成される）。

【0075】本発明の他の実施態様において、水平および垂直シンドロームは両方とも SRAM 15 内に記憶される。なぜなら、これらは、第 1 水平バスにおいて同時に生成されるからである。このように、上記の実施態様のように、次の水平バスの間に水平シンドロームを再生成する必要はない。シンドロームは、水平および垂直バスの間に単に取り出され、符号語を訂正するために使用される。本実施態様は、積符号を訂正するために多数の

21

パスが必要である場合（例えば、記録密度を増加させることによって、ECC冗長シンボルが減少する場合、またはSNRが減少する場合）に特に有利である。

【0076】図6Aおよび図6Bは、それぞれ、本実施態様による訂正値を用いて垂直および水平シンドロームを更新するための回路を示す。これらの回路は、オフセットを考慮する必要がないので、誤り訂正値21を、 α で乗算する必要がないこと以外は、図5の回路と実質的に同様に動作する。好ましい実施態様において、第1水平パスにおいて水平および垂直誤りシンドロームを最初に生成するための図4および図5の回路は、次のパスにおいて誤りシンドロームを更新するための図6Aおよび図6Bの回路と共用される。制御ライン30を介したアドレッシングは、どの誤りシンドロームのセット（水平または垂直）が、適切な時点でSRAM15から取り出されるかを決定する。

【0077】図7Aは、本発明の動作を説明するフロー図であり、図示しないコントローラによって実行される。第1の水平パス37の間に、水平および垂直ECCシンドロームおよびデータCRCシンドロームが同時に生成され、SRAM15内に記憶される。また、第1の水平パス37の間に、水平符号化語が訂正され、訂正値を用いて、垂直シンドロームおよびSRAM15内に記憶された誤りCRCシンドロームが更新される。第1のおよび次の水平パスの後、ステップ38で、積符号への訂正の有効性および完全性が最終CRCシンドロームを用いて確認される。水平パス後も誤りが残る場合は、ステップ40で垂直パスが実行され、垂直符号語が訂正される。ここでは、訂正値を用いて、シンドロームバッファ内に記憶された水平シンドロームが更新される。垂直パスの後、ステップ42で、訂正の有効性および完全性が最終CRCシンドロームを用いて確認される。垂直パス後も誤りが残る場合は、ステップ44で別の水平パスが実行され、水平符号語が訂正される。実際の適用に依存して、訂正はSRAM15内に記憶された水平シンドロームを用いて行われる（すなわち、水平シンドロームはSRAM15内で既に利用可能な場合は再生成されない）。水平および垂直パス、ならびにCRC検査は、積符号が訂正されるかまたは訂正不能と決定されるまで繰り返される。

【0078】第1の水平パスの間に水平および垂直シンドロームならびにデータCRCシンドロームを同時に生成するフロー図を図7Bに示し、また、第1のパスの間に水平符号語を訂正し、訂正値を用いて垂直シンドロームおよび誤りCRCシンドロームを更新するフロー図を図7Cに示す（図7Bおよび図7Cのフロー図は平行して実行される）。図7Bを参照して、ステップ46で、COLおよびROW変数がゼロに初期化される。SRAM15がクリアされ、ブロック訂正不能誤りフラグ（BLK_UNC）がクリアされ、そしてFIRST_PA

22

SSフラグが設定されて、これが第1の水平パスであることを示す。次に、ステップ48で、ROWの水平符号語に対してシンボルが読み出され、これを用いて、上述のように、図4の回路を用いて水平ECCシンドロームが、および図5の回路を用いて垂直ECCシンドロームが更新される。ステップ50でFIRST_PASSフラグが設定されている場合は、このデータシンボルはまた、図7Fのフロー図を実行することによってステップ52でデータCRCシンドロームを更新するためにも用いられる。次にステップ54でCOL変数がインクリメントされ、現在の水平符号語のための次のシンボルを用いて、第1の水平パスの間に水平および垂直ECCシンドロームならびにデータCRCシンドロームが更新される。

【0079】ステップ56でCOL変数が182に等しいときは、現在の水平符号語のための最終シンボルが読み出されたことを意味する。ステップ58でループを実行して、誤り訂正手順（図7Cのフロー図）が前の水平符号語の処理を終了させるのを待つ。前の符号語の訂正が終了すると、図7Cのステップ88で、訂正フラグが非使用中にリセットされる。図7Bのステップ60で訂正フラグは使用中に設定され、C_ROW変数は現在のROWに設定され、ROW変数はインクリメントされ、COL変数はゼロにリセットされる。この時点で、図7Cの訂正手順が実行され、現在の水平符号語（すなわちC_ROWの符号語）が訂正され、同時に、次の水平符号語（すなわちROWの符号語）のためにECCシンドロームが生成される。

【0080】図7Cを参照して、ステップ64でループを実行して、図7Bのシンドローム生成手順が、現在の水平符号語の処理を終了して訂正フラグを使用中に設定するのを待つ。ステップ66で、訂正の間に現在のコラムを追跡するC_COL変数がゼロにリセットされ（すなわち、訂正中の水平符号語の最初のシンボルにリセットされ）、符号語訂正不能フラグ（CW_UNC）がクリアされる。ステップ68で、訂正中の水平符号語のためのECCシンドロームがSRAM15から取り出され、誤りロケータ多項式が生成され、ルートが有効シンボル位置に対応するかどうか決定される。ステップ69で、ECCシンドロームが、誤りがないことを示すゼロである場合は、ステップ71で誤りCRCシンドロームが更新される。ステップ70で、シンドロームが、誤りが多すぎるか、または誤りロケータ多項式のルートが無効シンボルを指す場合は、ステップ73で、CW_UNCおよびBLK_UNCフラグが設定され、符号語およびブロック全体（積符号）が訂正不能誤りを含むことを示し、ステップ71で誤りCRCシンドロームが更新される。ステップ70で符号語が訂正可能である場合は、ステップ72で、C_ROWの水平符号語のためのECCシンドロームがクリアされる（ゼロにセットされ

23

る)。次にループを実行して、C__ROWの水平符号語が訂正される。ステップ74で、C__COLのシンボルが誤りである場合は、分岐を実行して、データバッファ内に記憶されたシンボルが訂正され、訂正値を用いて垂直シンドロームが更新される。この訂正は、ステップ76でC__COLがCOLより小さくなるまで遅延される。すなわち、図7Cの誤り訂正手順は、図7Bのシンドローム生成手順が現在の訂正コラムC__COLをパスするまで待つ。これは、図5の回路が正しく作動するためには必要である。何故なら、この回路は、訂正値21に α^{133} を乗算して、1つのシンボルのオフセットを計算するものであるからである。図7Cのステップ78で、データバッファ内に記憶された水平符号語シンボルが訂正値を用いて訂正され、ステップ80で、訂正値を用いて、図5に関連して上述したようにC__COLの垂直符号語のための垂直シンドロームが更新される。ステップ81で、誤りCRCシンドロームが訂正値により更新され（訂正値がゼロの場合でも）、ステップ82で、変数C__COLが水平符号語の次のシンボルにインクリメントされ、訂正ループが再実行される。

【0081】ステップ84でC__COLが182に等しい場合は、水平符号語の最終シンボルが処理されたことを意味する。次の垂直パスの間に現在の水平符号語のための誤りシンドロームの更新を迅速に行うために、ステップ86で、これらの誤りシンドロームが最初の符号語シンボルに再配置される。これは次の計算によって実行される。

【0082】

【数10】

$$SYN_1 = SYN_1 \cdot x^{-182} \text{ MOD } (x + \alpha^1).$$

上記の動作を行う回路について以下により詳細に述べる。

【0083】図7Cのステップ88で、訂正使用中フラグがクリアされ、ステップ64で、訂正手順は図7Bのシンドローム生成手順が次の水平符号語のための誤りシンドロームの生成を終了するのを待つ。シンドローム生成手順および誤り訂正手順は、最終水平符号語が処理される（すなわち、図7Bのステップ62でROWが208に等しくなる）まで平行して実行される。このとき、ステップ63でFIRST_PASSフラグがクリアされ、制御は図7Aに戻る。図7Aのステップ38で、第1の水平パス後も誤りが残っている場合は、ステップ40で垂直パスが実行される。この手順のフロー図は図7Dに示す。

【0084】ステップ90で、訂正変数C__COLおよびC__ROWはゼロにリセットされ（すなわち、第1の垂直符号語の最初のシンボルにリセットされ）、符号語訂正不能フラグ(CW__UNC)はクリアされる。次にステップ92で、訂正中の垂直符号語のためのシンドロ

24

ームがSRAM15から取り出され、誤りロケータ多項式を生成し、ルートが有効シンボル位置に対応するかどうかを決定する。ステップ93で、ECCシンドロームがゼロで誤りが無いことを示す場合は、ステップ95で誤りCRCシンドロームが更新される。ステップ94で、シンドロームが、誤りが多すぎるか、または誤りロケータ多項式のルートが無効シンボルを指す場合は、ステップ97で、CW__UNCおよびBLK__UNCフラグが設定され、ステップ95で誤りCRCシンドロームが更新される。ステップ94で、符号語が訂正可能である場合は、ステップ96で、垂直符号語のためのECCシンドロームがクリアされる（ゼロに設定される）。次にループを実行して、C__COLの垂直符号語が訂正される。ステップ98で、C__ROWのシンボルが誤りである場合は、分岐を実行して、ステップ100でデータバッファ内に記憶されたシンボルが訂正され、ステップ102で訂正値を用いて水平シンドロームが更新される。図6Aの回路を用いて、図5を参照して上述したのと類似の方法で水平シンドロームが更新される。C__ROWの水平ECCシンドロームがSRAM15から取り出され、 α^{133} で乗算され、訂正値35iに加算され、SRAM15内に戻される。図7Dのフローチャートには示していないが、訂正が行われない場合も（すなわち訂正値がゼロの場合も）符号語の各シンボルに対して水平シンドロームが更新される。これにより、図6Aのシンドローム更新回路が簡略化される。すなわち、ECCシンドロームを次の符号語シンボルに位置付けるのに、 α^1 乗算器33iしか必要としない。

【0085】図7Dのステップ104で、誤りCRCシンドロームが訂正値に更新され（訂正値がゼロの場合も）、ステップ105で、C__ROW変数が現在の垂直符号語の次のシンボルにインクリメントされる。ステップ106で、C__ROWが208に等しくなる、つまり現在の垂直符号語の最終シンボルが処理されるまで、訂正ループが繰り返される。最終シンボルが処理されると（または、ステップ94で符号語が訂正不能である場合）、ステップ107で、C__COLの垂直符号語のための誤りシンドロームが符号語の最初のシンボルに再配置される。これは次の計算によって実行される。

【0086】

【数11】

$$SYN_1 = SYN_1 \cdot x^{-208} \text{ MOD } (x + \alpha^1).$$

上記の動作を行う回路について以下により詳細に述べる。

【0087】ステップ108で、C__COL変数が次の垂直符号語にインクリメントされ、C__ROW変数がゼロにリセットされて、次の垂直符号語の最初のシンボルを指す。ステップ110でC__COLが182に等しくなり、垂直符号語のすべてが処理されたことを示すまで、図7Dの訂正手順が繰り返される。

25

【0088】垂直バスの最後に、ステップ42で積符号に誤りが残っている場合は、ステップ44で水平訂正バスが実行される。水平シンドロームがSRAM15内に記憶されていない場合、ステップ44で図7Bのフロー図が実行され、水平ECCシンドロームが再生成される(FIRST_PASSを偽として)。しかし、SRAM15内に水平ECCシンドロームが記憶されている場合は、図7Eのフロー図が実行され、単に訂正システムの待ち時間を低減させるECCシンドロームが取り出され処理される。

【0089】図7Eのステップ112で、訂正変数C__COLおよびC__ROWがゼロにリセットされ(すなわち、第1の水平符号語の最初のシンボルにリセットされ)、符号語訂正不能フラグ(CW__UNC)がクリアされる。次にステップ114で、訂正中の水平符号語のためのシンドロームがSRAM15から取り出され、誤りロケータ多項式を生成し、ルートが有効シンボル位置に対応するかどうかを決定する。ステップ115で、ECCシンドロームがゼロで誤りがないことを示す場合は、ステップ117で誤りCRCシンドロームが更新される。ステップ116で、シンドロームが、誤りが多すぎるか、または誤りロケータ多項式のルートが無効シンボルを指す場合は、ステップ119でCW__UNCおよびBLK__UNCフラグが設定され、符号語およびブロック全体(積符号)が訂正不能誤りを含むことを示し、ステップ117で、誤りCRCシンドロームが更新される。ステップ116で、符号語が訂正可能である場合は、ステップ118で、C__ROWの水平符号語のためのECCシンドロームがクリアされる(ゼロに設定される)。次にループを実行して、C__ROWの水平符号語が訂正される。

【0090】ステップ120で、C__COLのシンボルが誤りである場合は、分岐を実行して、ステップ122でデータバッファ内に記憶されたシンボルが訂正され、ステップ124で訂正値を用いて垂直シンドロームが更新される。垂直バスの間に水平ECCシンドロームを更新するのに図6Aを参照して上述したのと同様の方法で、図6Bの回路を用いて垂直ECCシンドロームが更新される。C__COLの垂直ECCシンドロームがSRAM15から取り出され、 $a^i 33_i$ で乗算され、訂正値 35_i に加算され、SRAM15内に戻される。図7Eのフローチャートには示していないが、訂正が行われない場合も(すなわち訂正値がゼロの場合も)符号語の各シンボルに対して、垂直ECCシンドロームが更新される。これにより図6Bのシンドローム更新回路が簡略化される。すなわち、ECCシンドロームを次の符号語シンボルに位置付けるのに、 a^i 乗算器 33_i しか必要としない。

【0091】図7Eのステップ126で、誤りCRCが訂正値を用いて更新され(訂正値がゼロの場合も)、ス

26

テップ127で、C__COL変数が現在の水平符号語の次のシンボルにインクリメントされる。ステップ128で、C__COLが182に等しい場合は、現在の水平符号語の最終シンボルが処理されたことを意味する。最終シンボルが処理されると(または、ステップ116で符号語が訂正不能である場合)、ステップ130で、C__ROWの水平符号語のための誤りシンドロームが符号語の最初のシンボルに再配置される。これは次の計算によって実行される。

10 【0092】

【数12】

$$SYN_i = SYN_i - x^{-182} \text{ MOD } (x + \alpha^4).$$

ステップ132で、C__ROW変数が次の水平符号語にインクリメントされ、C__COL変数がゼロにリセットされて、次の水平符号語の最初のシンボルを指す。ステップ134でC__ROWが208に等しくなり、水平符号語のすべてが処理されたことを示すまで、図7Eの訂正手順が繰り返される。

【0093】再び図7Aを参照すると、ステップ44で水平バスが完了した後、誤りが残っている場合は、ステップ40で別の垂直バスが実行される。積符号が訂正されるかまたは訂正不能であると決定されるまで、繰り返し水平および垂直バスが続けられる。各データセクタのためのデータCRCシンドロームおよび誤りCRCシンドロームを生成するフロー図を図7Gから図7Iに示し、CRCシンドロームを生成する回路を図8から図11に示す。これらのフロー図および回路は以下のように動作する。

【0094】[CRCシンドロームの生成および検証]図3Aの各データセクタのためのCRCシンドロームは2つの部分、すなわちデータCRCシンドロームおよび誤りCRCシンドロームで生成される。データCRCシンドロームは、非訂正データに対するCRCとして生成され、誤りCRCシンドロームは、訂正値に対するCRCとして生成される。図3Aの積符号のデータセクタの処理が終了すると、データCRCシンドロームと誤りCRCシンドロームとが組み合わされて最終CRCシンドロームが生成され、定数と比較されて、このセクタに対する訂正が有効且つ完全であるかどうか決定される。上述のように、データCRCシンドロームおよび誤りCRCシンドロームは、ランダム化データに対して積符号を訂正すると同時に生成される。すなわち、従来のように訂正後およびデータの非ランダム化後に生成されるのではない。

【0095】図1の従来の誤り訂正システムでは、CRC冗長シンボルCRC_{red}は、データ多項式D(x)を生成器多項式G(x)で割った剰余除算として生成される。

【0096】

【数13】

50

27

$$CRC_{RED} = D(x) \cdot x^{n-k} \bmod G(x).$$

CRC冗長シンボルCRC_{RED}をデータ多項式D(x)に加算した後、疑似ランダムデータシーケンス多項式R(x)を加算することによってデータをランダム化し、この結果、符号語多項式C(x)がディスクに書き込まれる。

【0097】

【数14】

$$C(x) = (D(x) \cdot x^{n-k} + CRC_{RED}) + R(x).$$

読み戻されると、受け取られた符号語多項式C'(x)はECC冗長シンボルを用いて訂正され、訂正された符号語から疑似ランダムデータシーケンスR(x)が減算され、以下のCRCシンドロームS_{CRC}が生成される。

【0098】

【数15】

$$S_{CRC} = (C'(x) - R(x)) \bmod G(x).$$

CRCシンドロームS_{CRC}は次にゼロと比較され、訂正の有効性および完全性が確認される。

【0099】本発明では、CRC検査は、訂正された符号語を非ランダム化する前に行われる。本発明の実現可 *20

$$C'(x) \bmod G(x) = (D(x) \cdot x^{n-k} + CRC_{RED} + R(x) + E(x)) \bmod G(x)$$

ここで、E(x)は誤り多項式である。上記の式は以下のように書き換えることができる。

$$C'(x) \bmod G(x) = (D(x) \cdot x^{n-k} + CRC_{RED}) \bmod G(x) +$$

$$E(x) \bmod G(x) +$$

$$R(x) \bmod G(x).$$

上記の式で、(D(x) · x^{n-k} + CRC_{RED}) mod G(x) = 0である。従って、以下の式となる。 ★ 【0103】

★ 【数19】

$$C'(x) \bmod G(x) = (E(x) \bmod G(x)) + (R(x) \bmod G(x)).$$

このように、受け取られた符号語C'(x)がECC冗長を用いて完全に訂正された場合(すなわち、E(x) = 0)、受け取られた符号語C'(x)に対して生成される最後のCRCシンドロームは、上記の式から分かるように、疑似ランダムデータシーケンスR(x)をCRC生成器多項式G(x)で除算した剰余除算に等しい。疑似ランダムデータシーケンスR(x)および生成器多項式G(x)は既知であるため、疑似ランダムデータシーケンスR(x)を生成器多項式G(x)で割った剰余は定数となる。すなわち、最後のCRCシンドロームは単にこの定数と比較され、ECCシンドロームを用いた訂正が有効かつ完全であるかどうか決定される。CRCシンドロームは、受け取られた符号語の訂正と同時に生成されるため、CRCシンドロームは直ちに利用可能となる。つまり、従来技術のようにCRCシンドロームを生成するために、データバッファから符号語全体を読み出す必要はない。

【0104】CRCシンドロームを受け取られた符号語の訂正と同時に生成するために、本発明は、第1の水平バスの間に非訂正データに対してデータCRCシンドロ

28

*能な改変として、データをランダム化/非ランダム化するために用いられる疑似ランダムデータシーケンスにわたって、最後のCRCシンドロームをCRCと比較してもよい。これは、以下の数学関係により理解される。

【0100】

【数16】

$$CRC_{RED} = D(x) \cdot x^{n-k} \bmod G(x)$$

$$C(x) = (D(x) \cdot x^{n-k} + CRC_{RED}) + R(x)$$

ここで、D(x)はデータ多項式、CRC_{RED}はCRC冗長度、およびR(x)は、データをランダム化するためにデータ多項式に加えられる疑似ランダムデータシーケンス多項式である。結果としてディスクに書き込まれる符号語C(x)は、上述の従来技術の場合と同じである。しかし、データを非ランダム化する前に(すなわち、疑似ランダムデータシーケンス多項式R(x)を減算する前に)受け取られた符号語C'(x)をCRC生成器多項式で除算すれば、以下の関係が導かれる。

【0101】

【数17】

※ 【0102】

※ 【数18】

★ 【0103】

★ 【数19】

ームを生成し、また、水平および垂直バス両方の間に訂正值に対して誤りCRCシンドロームを生成する。図3Aの積符号のデータセクタの処理が終了すると、データCRCシンドロームと誤りCRCシンドロームとが組み合わされて最終CRCシンドロームを生成し、これが定数(R(x) mod G(x))と比較される。

【0105】本発明では、データCRCシンドロームおよび誤りCRCシンドロームは従来の方法では(すなわち、リニアフィードバックシフトレジスタ(LFSR)を用いて)生成されない。何故なら、垂直バスの間にデータシンボルがデータバッファから順次読み出されるのではないからである。そうではなく、データおよびCRCシンドロームが、処理中のデータシンボルの位置に対応するように調整される。この調整は、データCRCシンドロームおよび誤りCRCシンドロームを以下の値で乗算することによって実行される。

【0106】

【数20】

$$x^{k-8} \bmod G(x)$$

50 ここで、kは符号語にわたってシンドロームを「移動さ

29

せる」シンボルの数を表す。例えば、第1の水平バスの間にデータCRCシンドロームを生成するとき、データCRCシンドロームは、

【0107】

【数21】

$$x^{1 \cdot 8} \bmod G(x)$$

で乗算され、これにより、データCRCシンドロームが現在の水平符号語の次のシンボルに調整される。垂直バスの間、誤りCRCシンドロームは、

【0108】

【数22】

$$x^{1 \cdot 8 \cdot 8} \bmod G(x)$$

で乗算され、これにより、誤りCRCシンドロームが現在の垂直符号語の次のシンボルに調整される。CRCシンドロームを生成する方法の数学的な基礎について、CRC生成器および訂正検証器を実現する回路に関連して以下に述べる。

【0109】図3Aに示すように、16個のデータセクタがあり、それぞれにCRC冗長が付けられている。従って、第1の水平バスの間に16個のデータCRCシンドロームが生成され、水平および垂直バス両方の間に16個の誤りCRCシンドロームが更新される。これらのシンドロームは好ましくはSRAM15内に記憶され、これによりSRAM15を用いて、CD-ROMフォーマットのためのCIRC誤り訂正およびDVDフォーマットのためのCRCシンドローム生成の両方が可能になる。第1の水平バスの間に新しいセクタがそれぞれ処理されるに従って、現在のデータCRCシンドロームは開始値に初期化される。データCRCシンドロームを次のセクタのための開始値に初期化する前に、前のデータセクタのための現在のデータCRCシンドロームがSRAM15内に記憶される。同様に、垂直符号語の処理時には、誤り訂正システムが16個のデータセクタを通して垂直方向に進行するに従って、現在のデータセクタに対応する適切な誤りCRCシンドロームがSRAM15から取り出される。処理中の現在のデータセクタに従ってデータCRCシンドロームおよび誤りCRCシンドロームを更新するこのプロセスは、図7Fから図7Jのフロー図により理解される。

【0110】図7Fは、図3Aの積符号に対する第1の水平バスの間にデータCRCシンドロームを生成するフロー図を示す。この積符号の最初の2つのデータセクタは図3Bに示されている。図7Fのフロー図は、図7Bのステップ52で新しいデータシンボルがデータバッファから読み出される度に実行される。図7Fのステップ136で、検査を行ってデータCRCシンドロームを次のデータセクタの最初のシンボルに初期化すべきかどうかが決定される。例えば、第1の水平符号語の最初のシンボルを処理するとき、ステップ138で、データCRCシンドロームを記憶するデータレジスタDATA_R

30

EGが、図3Bの第1のデータセクタの最初のシンボル160により初期化される。次のデータセクタに到達すると（すなわち、ステップ136でROWが12に等しいとき）、ステップ138で第1のデータセクタのためのデータCRCシンドロームがSRAM15内に保存され、DATA_REGが、次の水平符号語のための最初のデータシンボル204により初期化される。

【0111】図7Fのフロー図について続けて説明すると、データセクタの最初のシンボルが処理されず、且つステップ144でCOLが172より小さい場合、ステップ146でデータCRCシンドロームは1シンボル

(1列)だけ右に調整され、現在のデータシンボルがデータCRCシンドロームに加えられる。ステップ144で、COLが171より大きい場合は、現在のデータシンボルはデータCRCシンドロームに加えられない。何故なら、これはCRC符号語には含まれないECC冗長シンボルであるからである（ECC冗長は、上述のように書き込み動作中にCRC符号語が生成された後で加えられる）。例えば、図3Bの第1の水平符号語が処理されるとき、データCRCシンドロームの生成は、図7Bのステップ60でCOLがゼロにリセットされて、次の水平符号語の処理が開始されるまで、シンボル148で停止状態となる。次に、ステップ146で、第2の水平符号語の最初のデータシンボル150をデータCRCシンドロームに加える前に、データCRCシンドロームは先ず1シンボルだけ右に（すなわち、図3Bのシンボル150に）調整される。

【0112】データセクタの最終水平符号語の処理が終了すると（すなわち、ステップ154でROW+1 mod 12が0に等しく、ステップ156でCOLが181に等しいとき）、データCRCシンドロームは、対応するデータセクタの最終シンボルに位置付けられる。例えば、第1のデータセクタの処理が終了すると、データCRCシンドロームは図3Bのシンボル152に対して位置付けられる。上述のように、それぞれの対応する誤りCRCシンドロームもまた、CRC検査を行う前にデータセクタの最終データシンボルに位置付けられる。

【0113】第1のデータセクタの処理が終了すると、図7Fのステップ136で、ROWは12に等しく、ROW mod 12はゼロに等しい。従って、ステップ138で、第1のデータセクタのための現在のデータCRCシンドロームはSRAM15内に保存され、DATA_REGは、SRAM15から取り出される、第2のデータセクタのための開始位置により（すなわち、図3Bのシンボル142に）初期化される。ステップ146で、第2のデータセクタのCRC符号語のためのデータシンボルがデータCRCシンドロームに加えられる。このプロセスは、16個のデータCRCシンドロームのすべてが生成されSRAM15内に記憶されるまで続く。

【0114】水平および垂直バスの間に誤りCRCシ

31

ドローンを生成するフロー図を図7 Gから図7 J]に示す。第1の水平バスの間に、図7 Cのステップ81で誤りCRCシンドロームが訂正值に更新される(訂正值がゼロの場合も)。図7 Gのステップ159で、現在のバスが水平であるか垂直であるかに依存して分岐が実行される。現在水平バスを行っている場合は、ステップ161でC_ROWが191より大きいならば、ECC冗長はCRCシンドロームの一部ではないので、誤りCRCシンドロームは更新されない。垂直バスの場合は、ステップ162で訂正列C_COLおよび訂正行C_ROW mod 12がゼロであるならば、ステップ164で、誤りCRCシンドロームERR_REGを生成するレジスタに、現在のデータセクタに対応するSRAM15からの一部誤りCRCシンドロームがロードされる。

【0115】ステップ166で、符号語が訂正不能である(すなわち、訂正不能フラグ(CW_UNC)が設定される)か、または現在の水平符号語のためのECCシンドロームがゼロで訂正が不可能であることを示す場合は、ステップ168で、符号語が現在のデータセクタの最終水平符号語であるかどうか依存して分岐が実行される。データセクタの最終符号語である(すなわち、C_ROW+1 mod 12がゼロに等しい)場合は、ステップ169で、誤りCRCシンドロームを171シンボルだけ右にシフトすることによって、現在のデータセクタのための誤りCRCシンドロームが、CRC符号語の最終シンボルに(例えば、図3 Bのシンボル158からシンボル152に)位置付けられ、C_COLが172に設定される。後述の乗算テーブル数を減らすために、ステップ169での誤りCRCシンドロームの171シンボルだけ右へのシフトは、誤りCRCシンドロームを1シンボルだけ右にシフトすることを171回繰り返すループにおいて実行される。現在の符号語がデータセクタの最終符号語でない場合は、ステップ170で、誤りCRCシンドロームは、単に、現在の水平符号語を飛ばすために1行下に調整される。この後、制御は図7 Eに戻り、データセクタの次の水平符号語の処理を続ける。

【0116】ステップ166で、符号語訂正不能フラグ(CW_UNC)が設定されず、且つ現在の水平符号語のためのECCシンドロームがゼロでない場合は、図7 Hのステップ172で、現在の訂正列C_COLの値に基づいて分岐が実行される。C_COLが0から171である場合は、現在の水平符号語の最初のデータシンボルを処理していないならば(すなわち、ステップ174でC_COLが0でないならば)、ステップ176で誤りCRCシンドロームが1シンボル(1列)だけ右に調整され、ステップ178で現在のデータシンボルのための訂正值が誤りCRCシンドロームに加えられる。このプロセスは、現在の水平符号語のための訂正值のすべてが、誤りCRCシンドロームに加えられるまで続く。水

32

平符号語のECCシンボルは(ステップ172でC_COLが173から180であるときは)、上述のようにこれらはCRC符号語の一部ではないため、誤りCRCシンドロームには加えられない。現在の水平符号語の最終データシンボルが処理されると(すなわち、ステップ172でC_COLが172に等しいとき)、ステップ180で、誤りCRCシンドロームが、現在の水平符号語が現在のデータセクタの最終符号語であるかどうかに基づいて調整される。符号語がデータセクタの最終水平符号語でない場合は(すなわち、ステップ180で(Row+1) mod 12が0ではない場合は)、ステップ182で、誤りCRCシンドロームは1シンボルだけ右に調整され、これにより次の水平符号語の最初のシンボルに対して位置付けられる。現在の水平符号語が現在のデータセクタの最終符号語である場合は、ステップ181で、DATA_REGが現在のデータセクタのためのデータCRCシンドロームと共にSRAM15からロードされ、ERR_REGと組み合わせられる。上述のように結果が疑似ランダムシーケンスにわたってCRCに等しくない場合は、ステップ183でブロック訂正不能フラグ(BLK_UNCフラグ)が設定される。次に、ステップ184で、誤りCRCシンドロームは12行だけ上におよび1シンボルだけ右に調整されて、データセクタの最初のシンボルに対して再配置される。例えば、誤りCRCシンドロームが、図3 Bの第1のデータセクタのCRC符号語の最終シンボル152に位置する場合、ステップ184で誤りCRCシンドロームを12行だけ上におよび1シンボルだけ右に調整することにより、誤りCRCシンドロームはデータセクタの最初のシンボル160に位置付けられる。次にステップ185で、現在のデータセクタのための誤りCRCシンドローム(ERR_REGに記憶)がSRAM15内に再び記憶される。

【0117】図7 Gおよび図7 Hの誤りCRC更新手順は、図3 Aの16個のデータセクタすべてに対して誤りCRCシンドロームが生成され、疑似ランダムシーケンスにわたってCRCと比較されるまで繰り返される。第1の水平バスの終了時に、図7 Kのステップ254でBLK_UNCフラグが調べられ、積符号の訂正の有効性および完全性が確認される。誤りが残っている場合には、垂直バスが実行され、図7 Dのステップ104で(または、垂直符号語が飛ばされる場合はステップ95で)、図7 Iの誤りCRC更新手順が実行されて、誤りCRCシンドロームを、垂直符号語に適用された訂正值により更新する。

【0118】図7 Gのステップ159で、制御は図7 Iに分岐して、垂直バスのための誤りCRCシンドロームを更新する。図7 Iのステップ186で現在の訂正列C_COLが171より大きい場合は、水平ECCシンボルに対する垂直符号語はCRC符号語に含まれない

め、何も更新されない。ステップ186でC__COLが172より小さい場合は、ステップ188で検査が行われ、誤りCRCシンドロームが、現在の垂直符号語を下向きに進行して現在のデータセクタの最終行に到達したかどうかを決定する。現在の訂正行C__ROW mod 12がゼロに等しい場合は、ステップ190で、現在のデータセクタのための一部誤りCRCシンドロームがSRAM15から取り出され、ERR__REGにロードされる。

【0119】ステップ192で符号語訂正不能誤りフラグ(CW__UNC)が設定されているか、または現在の垂直符号語のためのECCシンドロームがゼロである場合は、図7Jのフロー図が実行されて、誤りCRCシンドロームを1シンボルだけ右に調整する(すなわち、現在の垂直符号語が飛ばされる)。ステップ193でC__COLが171に等しくない場合は(すなわち、データセクタの最終垂直符号語を処理していない場合は)、ステップ194で、第1のデータセクタのための誤りCRCシンドロームが1シンボル(1列)だけ右に調整される。ステップ196で現在のデータセクタのための一部誤りCRCシンドロームがSRAM15内に記憶され、ステップ197でC__ROWが12だけインクリメントされる。すなわち次のデータセクタに移される。ステップ198でC__ROWが191より大きくない場合は、ステップ200でERR__REGが次のデータセクタのための一部誤りCRCシンドロームと共にSRAM15からロードされ、ステップ194でこの誤りCRCシンドロームが1シンボルだけ右に調整され、ステップ196でSRAM15内に再記憶される。このループは、ステップ198でC__ROWが191より大きくなるまで繰り返され、191より大きくなると、すべての誤りCRCシンドロームが1シンボル(1列)だけ右に調整されて、現在の垂直符号語が飛ばされる。

【0120】図7Jのステップ193でC__COLが171に等しい場合は、最終垂直符号語(ECC冗長の前)は飛ばされることになる。従って、各データセクタのための最終CRCシンドロームを各データセクタのために生成し、誤りが残っている場合はBLK__UNCフラグを設定する必要がある。ステップ201で、現在のデータセクタ(ERR__REGに記憶)のための誤りCRCシンドロームが11データシンボルだけ下に(例えば、図3Bのシンボル148からシンボル152に)調整される。ステップ201は実際には、ERR__REGを1シンボル(D1)だけ下に調整するのを11回繰り返すループとして実現される。次にステップ203で、DATA__REGが、SRAM15からの現在のデータセクタのためのデータCRCシンドロームにより初期化され、DATA__REGをERR__REGと組み合わせることによって最終CRCシンドロームが生成される。最終CRCシンドロームが疑似ランダムシーケンスにわ

たってCRCに等しくない場合は、ステップ205でブロック訂正不能フラグ(BLK__UNC)が設定される。ステップ207で、誤りCRCシンドロームは、12行だけ上におよび1データシンボルだけ右に(例えば、図3Bのシンボル152からシンボル160に)調整することによって、データセクタの開始点に調整される。ステップ200で次のデータセクタのための誤りCRCシンドロームがERR__REGにロードされ、上記の手順が繰り返される。ステップ198でC__ROWが191より大きい場合は、各データセクタに対してCRC検査が行われ、誤りCRCシンドロームは各データセクタの開始点に再配置されているはずである。

【0121】再び図7Iを参照して、ステップ192で、符号語訂正不能フラグ(CW__UNC)がセットされず、ECCシンドロームがゼロでなければ、ステップ206で分岐が実行され、現在の訂正行C__ROWの値に基づいて誤りCRCシンドロームが更新される。ステップ206でC__ROWが192~207であれば、CRC符号語にECC冗長記号が含まれないため、制御は単に戻るだけである。C__ROWが0~191であれば、現在の誤りCRCシンドロームは、ステップ208で1記号だけ下に調整され、ステップ210で、次の記号の訂正値が誤りCRCシンドロームに加えられる。誤りCRCシンドロームがデータセクタの最初の行にある場合(即ち、ステップ212でC__ROW mod 12がゼロである場合)、誤りCRCシンドロームを1記号だけ下に調整するステップ208はスキップされる。

【0122】ステップ209で、現在のデータセクタの最後の行に達すると(即ち、C__ROW+1 mod 12がゼロであれば)、誤りCRCシンドロームは、垂直方向の次の符号語のデータセクタのトップ位置に調整される。これは、ステップ211で誤りCRCシンドロームを1記号下(D1)に調整し、さらに、ステップ213で1記号右で12記号上(UP12__R1)に調整する。ステップ215でC__COLが171に等しければ、誤りCRCシンドロームは、データセクタの最後の記号の上に配置され、CRCチェックが行われる。ステップ217で、現在のデータセクタのデータCRCシンドロームとともに、DATA__REGがSRAM15からロードされ、ERR__REGと結合され、最終CRCシンドロームが生成される。最終CRCシンドロームが疑似ランダムシーケンス上のCRCに等しくなければ、ステップ219でブロック訂正不能フラグ(BLK__UNC)がセットされる。その後、ステップ213で、誤りCRCシンドロームは、12行上で1データ記号右(UP12__R1)に動かされ、データセクタの最初の記号上に再配置され(例えば、図3Bの記号152から記号160に動かされ)、ステップ214で、現在のデータセクタの誤りCRCシンドローム(ERR__REGに格納されている)が、SRAM15に復元される。

35

【0123】垂直方向のバスの終わりで、BLK_UNCフラグが図7Kのステップ254で検査され、積符号に対する訂正の有効性および完全性が検証される。誤りがまだ残っていれば、水平方向のバスがもう一度行われる。水平方向の符号語のECCシンドロームがSRAM15に格納されると、図7Eのフロー図が実行され、ステップ117および126で誤りCRCシンドロームが更新される。水平方向の符号語のECCシンドロームがSRAM15に格納されなければ、図7Cのフロー図が実行され、ステップ71および81で誤りCRCシンドロームが更新される。

【0124】図7Kのフロー図は、積符号が完全に訂正されているかどうか、または、水平方向もしくは垂直方向のバスの終わりで積符号が訂正不可能であるかどうかを判定するステップを示す。ステップ252で、システムは、ECC_BUSYフラグがクリアされるのを待ってから、BLK_UNCフラグの検査を行う。ステップ254でブロック訂正不能フラグ(BLK_UNC)がセットされず、前の水平方向または垂直方向のバスの間に訂正不可能な誤りに遭遇しなかったことが示されると、訂正手順は、従来技術で必要とされるような別のバスを行わずに、ステップ258で成功としてフローから出る。

【0125】ステップ254でBLK_UNCフラグがセットされると、ステップ256でパスカウント変数PASS_CNTがインクリメントされ、PASS_CNTが所定の最大値を上回ると、積符号は訂正不可能となり、訂正手順はステップ266で失敗としてフローから出る。ステップ256でPASS_CNTが所定の最大値未満であり、ステップ260で、前の水平方向および垂直方向のバスで変更がなされていなければ(訂正が行われていなければ)、それ以上バスを行っても役に立たないため、訂正手順はステップ266で再び失敗としてフローを出る。ステップ260で変更がなされていれば、ステップ262でBLK_UNCフラグがクリアされ、水平方向または垂直方向のバスをもう一度実行することによって、ステップ264で訂正手順が続けられる。

【0126】CRCシンドロームS_{CRC}が、積符号の訂正と同時に計算されるため、訂正手順は、水平方向または垂直方向のいずれかのバスの終わりでうまく終了し得る。従って、本発明では、訂正が終了したかどうかを検証するために、従来技術で必要とされていた別のバスが必要でない。さらに、本発明では、データを非ランダム化して訂正プロセスの終わりにCRCシンドロームS_{CRC}を生成するために必要とされていた別のバスが必要でない。従って、本発明は、光記憶装置のスループットを大幅に増加することにより、従来技術に対する大幅な改良を提供する。

【0127】[CRC生成器回路]図2のCRC生成器お

36

よび訂正検証器19は、書き込み動作の間に、図3Aに示される16個のデータセクタについてCRC冗長記号を生成し、読み出し動作の間に、図7A~図7Jを参照して上で説明したように誤り訂正器18によって行われた訂正を検証する際に用いられるCRCシンドロームS_{CRC}を生成する。図8は、書き込み動作の間にCRC冗長記号を生成するための従来のリニアフィードバックシフトレジスタ(LFSR)を示す。図8に示されるLFSRの動作は周知であり、このLFSRは、入力多項式D(x)を以下に示す生成多項式G(x)で割る。

【0128】

$$G(x) = g_i x^i + g_{i-1} x^{i-1} + \dots + g_1 x + g_0$$

入力多項式D(x)の係数は、LFSRを通してシリアルにシフトされる。ここで、シフト数は、入力多項式の次数に1を加えた数に等しい。剰余、即ち、CRC冗長は、シフトレジスタの最終状態である。図3Aに示されるデータセクタの各々についてCRC冗長を生成するために、データのkビットは、多項式P(x)の係数として表される。従って、CRC冗長は、以下のように計算される。

【0129】

$$\text{CRC冗長} = P(x) \cdot x^{n-k} \bmod G(x)$$

ここで、n-kはCRC冗長記号の数であり、G(x)は生成多項式である。最後のシフト後のレジスタの内容がCRC冗長であり、このCRC冗長はその後、ユーザデータに付加されて、CRC符号語を形成する。このCRC符号語は、積符号に組み込まれて、その後、ディスクに書き込まれる。

【0130】読み出し動作の間、ディスクから読み出されたデータが処理され、以下の式に従ってCRCシンドロームS_{CRC}が生成される。

【0131】

$$\text{CRCシンドローム } S_{\text{CRC}} = C'(x) \bmod G(x)$$

ここで、C'(x)は、ディスクから読み出されたCRC符号語(CRC冗長を含む)である。従来技術では、誤り訂正器14が訂正をし終わると、データバッファ1からデータが読み出され、非ランダム化器4によって非ランダム化される。非ランダム化されたデータはその後シリアルに処理されるため、図8の同じLFSR回路を用いて、上記式によりCRCシンドロームS_{CRC}を生成することができる。

【0132】本発明では、図7A~図7Jを参照して説明したように、CRCシンドロームS_{CRC}は、積符号の訂正と同時に生成される。従って、データが一連の連続するビットとして処理されないため、CRCシンドロームS_{CRC}を生成するために図8のLFSR回路を用いることはできない。本発明のCRCシンドローム生成器がどのようにしてCRCシンドロームS_{CRC}を生成するかを説明する前に、本発明のCRCシンドローム生成器の概要を説明する。

【0133】図9は、DATA_CRC回路300およびERROR_CRC回路302を含む、図2のCRCシンドローム生成器19のブロック図である。図7Aを参照して上で説明したように、DATA_CRCは、ステップ37で、図2のデータバッファ1から読み出された未訂正のランダム化されたデータを用いて、図3Aの積符号上の最初の水平方向のパスの間に生成される。ERROR_CRCは、水平方向および垂直方向の符号語の反復処理の間に誤り訂正器18によって生成される訂正值を用いて生成される。データセクタの終わりに達すると、DATA_CRCおよびERROR_CRCは、COMBINE回路304によって結合され、最終CRCシンドロームSCRC306が生成される。この最終CRCシンドロームSCRC306は、比較器307で定数と比較され、データセクタがまだ誤りを含んでいるかどうか判定される。COMBINE回路304によって行われる数学的機能は、DATA_CRCおよびERROR_CRCの単純な排他的OR(XOR)である。

【 0 1 3 4 】 上述のように、図 3 A の積符号には 1 6 個のデータセクタがある。したがって、SRAM 1 5 に格納されるデータ CRC シンドロームおよび誤り CRC シンドロームは 1 6 個である。CRC チェックは、SRAM 1 5 からデータ CRC シンドロームおよび誤り CRC シンドロームをそれぞれ読み出し、これらのデータ CRC シンドロームおよび誤り CRC シンドロームをそれぞれ DATA CRC 回路 3 0 0 および ERROR CRC 回路 3 0 2 にロードすることによって、各データセクタについて行われる。比較器 3 0 7 は、最終 CRC シンドローム S CRC を、各データセクタをランダム化するために用いられる疑似ランダムデータシーケンス上の CRC に等しい定数と比較する。疑似ランダムデータシーケンスはブロック（積符号）ごとに異なるため、記憶媒体から新しいブロックが読み出されるたびに、対応する CRC 定数が比較器 3 0 7 にロードされる。

$$\text{CRC シフトレジスタ} \quad S_{\text{CRC}} = \bar{c}_0(x) \cdot x^{8 \cdot 0} \bmod G(x) + \bar{c}_1(x) \cdot x^{8 \cdot 1} \bmod G(x) + \dots + \bar{c}_j(x) \cdot x^{8 \cdot j} \bmod G(x).$$

ここで、

【数 2 5】

$$\bar{c}_k(x)$$

は、符号語 $C'(x)$ からの 8 ビットの多項式である

(即ち、

【数 2 6】

$$\bar{C}_k(x) \cdot x^{r+m} \bmod G(x) = (\bar{C}_k(x) \cdot x^n \bmod G(x)) \cdot x^m \bmod G(x).$$

再び図 3 B を参照して、上記式を用いて、水平方向の符号語上の最初のパスの間の最初のデータセクタの最初の記号 160 についてのデータ CRC シンドロームが計算される。最初の記号 160 は、上記符号語 $C'(x)$ の最も有意な係数、および上記部分集合多項式 $C_j(x)$

*【0135】図9のDATA_{CRC}回路300およびERROR_{CRC}回路302は、図10により詳細に示される。図9のDATA_{CRC}回路300およびERROR_{CRC}回路302は、受け取ったCRC符号語多項式C'(x)を、以下のような複数の部分集合多項式の線形結合として表すことによって、データCRCシンドロームおよび誤りCRCシンドロームを生成する。

【0 1 3 6】

$$C'(x) = C_j(x) + C_{j-1}(x) + \dots + C_0(x)$$

ここで、各部分集合多項式 $C_k(x)$ は、符号語多項式 $C'(x)$ からの所定のビット数を含む。本明細書に開示される実施形態では、各部分集合多項式は、符号語多項式 $C'(x)$ の 8 ビットを含み、以下のように 16 進法で表される。

【 0 1 3 7 】

【数 2 3】

$$C_0(x) = 00000000 \dots 000000000000xx +$$

$$C_1(x) = 00000000 \dots 00000000 \infty 00 +$$

$$C_2(x) = 00000000 \dots 0000000x0000 +$$

 $\bullet \bullet \bullet +$

$$C_{j-1}(x) = 00x0000 \dots 000000000000 +$$

$$C_j(x) = x \times 000000 \dots 000000000000 =$$

$$C'(x) = xxxxxx \dots xxxxxx$$

このようにして、CRCシンドローム S_{CRC} は、概念的に、以下のように各部分集合多項式のCRCシンドロームの線形結合として生成され得る。

【0138】CRCシンドローム $Scrc = C_0(x) \bmod G(x) + C_1(x) \bmod G(x) + \dots + C_j(x) \bmod G(x)$ 上記式は、以下のように表すこともできる。

【0 1 3 9】

【数 2 4】

$$\bar{C}_k(x) = C_k(x) \cdot x^{-\partial_k}.$$

である)。本発明で用いられる別の数学的関係は、以下の式である。

【 0 1 4 0 】

【数 2 7】

のゼロでない係数を含む。最初の記号 160 は、(加算器 310 で、最上位ビットにゼロをパディング (zero padding) し、ゼロを加えた後に) 図 10 の 32 ビットレジスタ 308 にロードされる。その後、図 3B の符号語の次の記号 140 が読み出され、乗算器 312 で、レジ

39

スタ 308 の内容に $x^{K \bmod G(x)}$ が掛けられ、データ CRC シンドロームが 1 記号右にシフトされる（即ち、図 7 F のステップ 146 で $K=R1$ である）。その後、乗算結果は、（加算器 310 で符号語の次の記号 140 を加算して、その特定の部分集合多項式についての CRC シンドローム計算を開始した後に）レジスタ 308 に再ロードされる。この計算は、残りの記号に対して行われ、図 3 B の水平方向の最初の符号語の最後の記号 148 が読み出されるまで続けられる。その後、CRC シンドロームは、レジスタ 308 の内容に $x^{K \bmod G(x)}$ を掛けることによって、図 3 B の場所 150 に調整される。ここで、 K は、1 つ右の記号に等しい（即ち、上記図 7 F のステップ 146 で $K=R1$ である）。オフセット K の適切な値は、図 10 の乗算器 312 の SEL 制御ラインを介して選択される。

【0141】このプロセスは、図 3 B の最初のデータセクタの水平方向の最後の符号語の最後の記号が読み出されるまで続けられ、レジスタ 308 は、CRC 符号語 $C'(x)$ の最初の記号（即ち、部分集合多項式 $j(x)$ ）についてのデータ CRC シンドロームであって、その他の記号（即ち、その他の部分集合多項式）について計算されたデータ CRC シンドロームに加えられデータ CRC シンドロームを含み、これにより、最初のデータセクタの CRC 符号語 $C'(x)$ 全体についてのデータ CRC シンドロームを生成する。この時、最初のデータセクタについてのデータ CRC シンドロームは、図 3 B の最後の記号 152 に配置される。図 7 F のフロー図が再び実行され、次のデータセクタの最初の記号 204 が処理されると、図 7 F のステップ 138 で、最初のデータセクタについてのデータ CRC シンドロームが SRAM15 に格納され、次のデータセクタの最初の記号 204 がレジスタ 308 にロードされる。上記プロセスは、2 番目およびそれ以降のデータセクタについても繰り返され、最初の水平方向のバスの終わりで、16 個のデータ CRC シンドロームがすべて生成され、SRAM15 に格納される。

【0142】誤り CRC を生成するための図 9 の ERR ORCRC 回路 302 はまた、図 10 の回路を含む。図 2 の誤り訂正器 18 によって訂正値が生成されると、この訂正値は、加算器 310 でレジスタ 308 に加えられる。乗算器 312 は、水平方向または垂直方向の符号語の各記号が処理されているとき、訂正値が生成されるかどうかに関わらず（即ち、訂正値がゼロであっても）、レジスタ 308 の内容への適切な K オフセットの乗算を続ける。各データセクタの終わりで、データ CRC シンドロームおよび誤り CRC シンドロームが結合され、最終 CRC シンドローム $SCRCJ$ が生成される。上述のように、この最終 CRC シンドローム $SCRCJ$ は、訂正の有効であるかどうかおよび完全であるかどうかを判断するために用いられる。

40

【0143】図 10 の $x^{K \bmod G(x)}$ 乗算器 312 を実現するための好適な実施形態は、図 11 を参照して理解される。図 11 は、 $x^{K+i \bmod G(x)}$ という計算によって生成される剰余のテーブルを表す。ここで、 i は $\{0, \dots, 31\}$ に等しい。図 11 のテーブルは、CRC シンドロームの計算中に用いられる K オフセット値（即ち、 $R1$ 、 $D1$ および $UP12_R1$ ）の各々について生成される。その後、図 10 のレジスタ 308 の内容に適切なテーブルを掛けることによって（即ち、32 ビットベクトルに 32×32 行列を掛けることによって）乗算が行われる。

【0144】32 ビット CRC 生成多項式 $G(x)$ についての $x^{K \bmod G(x)}$ の乗算を実現するための実際のテーブルは、図 13 A ~ 図 13 E の VHDL ソースコードに示される。「constant r1_dvd_tbl」として示されたテーブルは、1 記号右へのシフト ($R1$) 調整を実現し、「constant d1_dvd_tbl」として示されたテーブルは、1 行下へのシフト ($D1$) 調整を実現し、「constant u12_r1_dvd_tbl」として示されたテーブルは、12 記号上、1 記号右のシフト ($UP12_R1$) 調整を実現する。

【0145】図 13 A ~ 図 13 E の VHDL ソースコードの残りは、図 10 のレジスタ 308 の内容に適切なテーブルを掛ける（即ち、32 ビットベクトルに 32×32 行列を掛ける）ことによって、実際の乗算を実行する。入力レジスタ即ちベクトルと行列との積が、出力ベクトルであり、出力ベクトルの各要素は、テーブル（即ち、行列）の i 番目の行の n 個の要素と、レジスタ（即ち、列入力ベクトル）の対応する成分との積の和を求めることによって生成される。この和は、以下のように表すことができる。

【0146】

【数 28】

$$y_i = \sum_{k=0}^{31} a_{ik} x_k$$

ここで、 y_i は、乗算器 312 の出力ベクトルであり、 a_{ik} は、図 11 のテーブルの i 番目の行の 32 ビットであり、 x_k は、図 10 のレジスタ 308 に格納された 32 ビットである。乗算器 312 からの出力ベクトル y_i は、加算器 310 で入力ビットに加えられ、その結果が、レジスタ 308 に格納される。

【0147】[SRAM] 図 2 の SRAM15 の構造および動作は、図 12 A を参照して理解される。上述のように、SRAM15 は、動作モードに依存して 2 つの機能を実行する。即ち、CD モードでは、SRAM15 は、 $C1/C2$ 符号化のためのデータバッファ処理を与え、DVD モードでは、SRAM15 は、積符号についての ECC シンドロームと、CRC 検証符号についての CR

41

Cシンドロームとを格納する。DVDモードの場合の好適な実施形態では、図3Aに示される積符号の垂直方向のECCシンドロームが182個だけ、16個のデータセクタに対応する16個のCRCシンドロームとともに、SRAM15に格納される。DVD積符号の208個の水平方向のECCシンドロームは、水平方向のバスの各々で再生成され、SRAM15には格納されない。当業者は、この構成が単に特定の実施形態であって、SRAM15の容量を、垂直方向および水平方向のECCシンドローム、およびCRCシンドロームを格納するように増加させることができることを認識するであろう。

【0148】図12Aを参照して、SRAM15は、好ましくは、16個の256×8ビットメモセル3140~31415のバンクとして実現される。各メモセルは、256データバイトにアクセスするための8ビット入力アドレスと、アドレス指定されたデータバイトを出力するための8ビット出力バスを含む。アドレス復号化器316は、制御ライン317を介して構成される動作モードに依存して、12ビットアドレス318を復号化する。CDモードの場合、メモセル3140~31415のバンクは、4k×8ビットバッファとしてアドレス指定される。即ち、アドレス318の12ビットを全部用いて、C1/C2復号化のための8ビットの1データバイトにアクセスする。アドレス318の最初の8ビットは、メモセル3140~31415の各々から同じデータバイトを選択するために用いられる。メモセルの出力は、トライステートバッファ3200~32015を介してワイアードORされる(wireored)。アドレス318の残りの4ビットを用いて、適切なトライステートバッファがイネーブルされ、それにより、バス322上に、C1/C2復号化に用いられる適切なデータバイトをアサートする。

【0149】DVDモードの場合、SRAM15は、256×128バッファとしてアドレス指定される。即ち、アドレス318の最初の8ビットだけを用いて、メモセル3140~31415の各々から同じデータバイトを選択する。各メモセルから出力される16データバイトは、324で結合され、図5および図6Bに示されるような垂直方向の符号語の16個のECCシンドロームを形成するか、または、図3Aに示されるデータセクタについてのCRCシンドロームの4バイトを形成する。

【0150】DVDモードの場合の垂直方向のECCシンドロームおよびCRCシンドロームの好適なマッピングは、図12Bに示される。最初の182個のアドレスが、182個の垂直方向の符号語の16個のECCシンドロームを格納するために用いられる。次の10個のアドレスはスキップされ、アドレス192~207は、図3AのDVD積符号の16個のデータセクタについての4バイトのデータCRCシンドロームおよび誤りCRC

42

シンドロームを16個格納するために用いられる。アドレス192~207を用いてデータCRCシンドロームおよび誤りCRCシンドロームにアクセスすることにより、アドレスの3つの最下位ビットしか変わらないため、複号化回路が簡略化される。SRAM15の残りは、DVDモードでは使用されない。

【0151】本発明の目的は、本明細書に開示された実施形態により十分に実現されている。当業者は、本発明の本質的な機能から逸脱することなく、様々な実施形態によって本発明の様々な局面を達成することができることを認識するであろう。例えば、図3Aに示される積符号は、典型的にはデジタルビデオディスク(DVD)で用いられるが、本発明は、コンパクトディスク(CD)に用いられるフォーマットを含む他の積符号フォーマットに等しく適用可能である。さらに、本発明は、積符号だけでなく、他の多次元符号にも適用され得る。このように、本明細書に開示された特定の実施形態は例示的なものであって、前掲の特許請求の範囲によって適切に解釈される本発明の範囲を限定することを意味するものではない。

【0152】

【発明の効果】本発明によれば、少なくとも以下の効果が得られる。

【0153】まず、ランダム化データに対してCRC検査を行うことによって、CRCシンドロームを生成する前にデータを非ランダム化するためのデータバッファへのアクセスに関連する待ち時間が避けられる。

【0154】さらに、PおよびQ符号語の訂正と同時に実行中にCRCシンドロームを生成することにより、CRC符号語を処理した直後に、訂正の有効性および完全性を検査するためにCRCシンドロームを利用可能にする。これにより、CRCシンドロームを生成するためにデータバッファにアクセスする必要がなくなる。

【図面の簡単な説明】

【図1】CD/DVD光記憶装置において典型的に用いられる従来の誤り訂正システムのブロック図である。

【図2】ランダムデータに対して実行中にCRCシンドロームを生成するCRC生成器および訂正検証器を有する、本発明の誤り訂正システムのブロック図である。

【図3A】16個のデータセクタを有するDVD光記憶装置において典型的に用いられる積符号のフォーマットを示す図である。

【図3B】図3Aの積符号の最初の2つのデータセクタのフォーマットを示す図である。

【図3C】本発明において用いられるデータランダム化器/非ランダム化器の詳細を示す図である。

【図3D】本発明において用いられるデータランダム化器/非ランダム化器の詳細を示す図である。

【図4】本発明の第1実施態様による水平符号語誤りシンドロームを生成する詳細な回路を示す図である。

43

【図5】水平符号語に対する誤りシンドロームの生成と同時に、垂直符号語誤りシンドロームを生成する詳細な回路を示す図である。

【図6A】本発明の他の実施態様について、シンドロームバッファが、水平および垂直符号語の両方に対する誤りシンドロームを記憶するとき、水平および垂直誤りシンドロームをそれぞれ更新する回路を示す図である。

【図6B】本発明の他の実施態様について、シンドロームバッファが、水平および垂直符号語の両方に対する誤りシンドロームを記憶するとき、水平および垂直誤りシンドロームをそれぞれ更新する回路を示す図である。

【図7A】本発明の誤り訂正システムによって実行されるステップの概略を示す流れ図である。

【図7B】水平符号語に対する第1パスの間に水平および垂直誤りシンドロームを同時に生成し、積符号に対する訂正の有効性および完全性を検査するためのCRCシンドロームを同時に生成するための流れ図である。

【図7C】第1（および次の）水平パスの間水平符号語を訂正し、訂正値を用いてシンドロームバッファおよびCRC誤りレジスタにおいて記憶される垂直誤りシンドロームを更新するための流れ図である。

【図7D】シンドロームバッファに記憶される垂直誤りシンドロームを用いて垂直パス中に垂直符号語を訂正し、訂正値を用いてシンドロームバッファおよびCRC誤りレジスタに記憶される水平誤りシンドロームを更新するための流れ図である。

【図7E】シンドロームバッファに記憶される水平誤りシンドロームを用いて次の水平パス中に水平符号語を訂正し、訂正値を用いてシンドロームバッファおよびCRC誤りレジスタに記憶される垂直誤りシンドロームを更新するための流れ図である。

【図7F】全積符号に対する第1水平パス中にデータCRCシンドロームを生成するための流れ図である。

【図7G】水平パス中に訂正値を用いて誤りCRCシンドロームを更新するための流れ図である。

【図7H】水平パス中に訂正値を用いて誤りCRCシンドロームを更新するための流れ図である。

【図7I】垂直パス中に訂正値を用いて誤りCRCシンドロームを更新するための流れ図である。

【図7J】垂直パス中に訂正値を用いて誤りCRCシンドロームを更新するための流れ図である。

【図7K】CRCデータレジスタの内容とCRC誤りレジスタの内容とを組合せ、水平および垂直パスの終わりにおける訂正の有効性および完全性を検査するための最終CRCシンドロームを生成する流れ図である。

【図8】書き込み動作中にCRC冗長を生成し、読出し動作中にCRCシンドロームを生成するのに用いられる従来のリニアフィードバックシフトレジスタ(LFSR)を示す図である。

【図9】CRCシンドロームのデータ部分を計算する

44

ためのDATA_{CRC}回路と、CRCシンドロームの誤り部分を計算するためのERROR_{CRC}回路と、DATA_{CRC}レジスタとERROR_{CRC}レジスタとを組合せて、ランダムデータパターンに対するCRCと等しい定数と比較される最終CRCシンドロームS_{CRC}を生成するための回路とを有するCRC訂正検証回路のブロック図である。

【図10】図9のDATA_{CRC}/ERROR_{CRC}回路の詳細なブロック図である。

【図11】乗算器が $x^k \text{ MOD } G(x)$ を計算するためのマトリクスの一般的な形態を示す図である。

【図12A】SRAMの構造、ならびにCDフォーマット用および積符号に対する部分シンドロームを記憶するためのC1/C2符号、およびDVDフォーマット用のCRC検証符号を復号化する間にどのようにSRAMが構成されるかを示す図である。

【図12B】本発明の好ましい実施態様における垂直ECCシンドロームおよびCRC検証シンドロームのSRAMマッピングを示す図である。

【図13A】32ビットCRC生成多項式 $G(x)$ についての $x^k \text{ mod } G(x)$ の乗算を実現するための実際のテーブルを生成するVHDLソースコードを示す図である。

【図13B】32ビットCRC生成多項式 $G(x)$ についての $x^k \text{ mod } G(x)$ の乗算を実現するための実際のテーブルを生成するVHDLソースコードを示す図である。

【図13C】32ビットCRC生成多項式 $G(x)$ についての $x^k \text{ mod } G(x)$ の乗算を実現するための実際のテーブルを生成するVHDLソースコードを示す図である。

【図13D】32ビットCRC生成多項式 $G(x)$ についての $x^k \text{ mod } G(x)$ の乗算を実現するための実際のテーブルを生成するVHDLソースコードを示す図である。

【図13E】32ビットCRC生成多項式 $G(x)$ についての $x^k \text{ mod } G(x)$ の乗算を実現するための実際のテーブルを生成するVHDLソースコードを示す図である。

【符号の説明】

- 1 データバッファ
- 3 ライン
- 4 ランダム化器/非ランダム化器
- 7 C1符号化器/復号化器
- 8 C2符号化器/復号化器
- 9 インターリーブ器/非インターリーブ器
- 15 SRAM
- 16 P/Q復号化器
- 17 ECC/シンドローム生成器
- 18 誤り訂正器

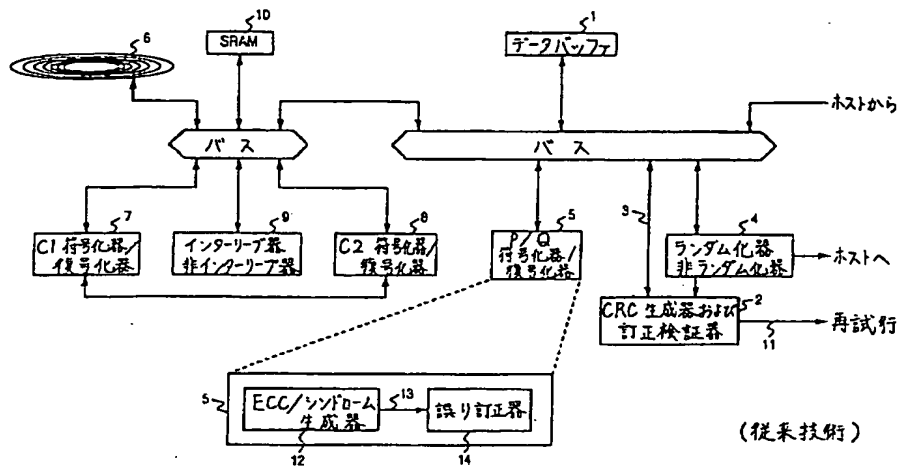
45

46

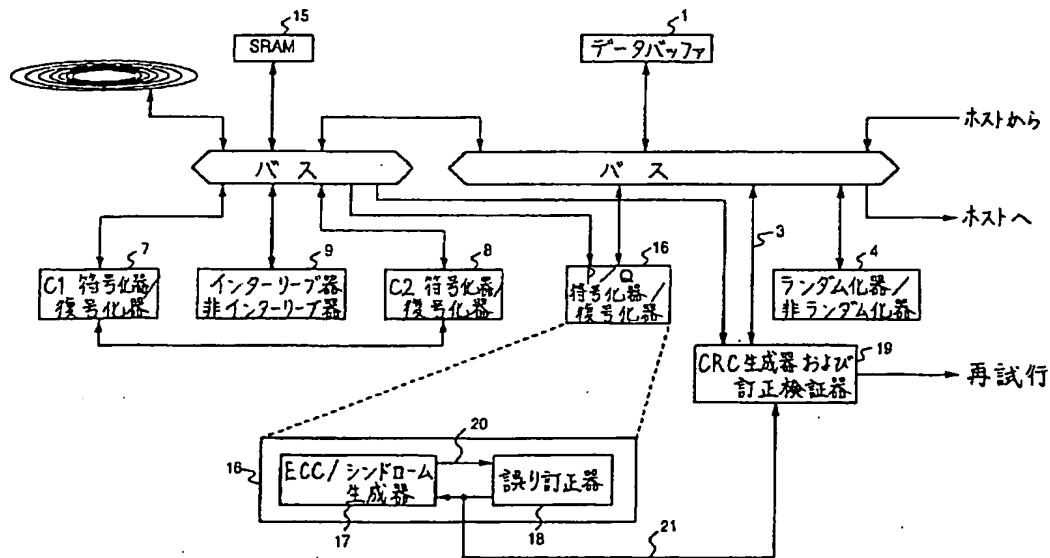
19 CRC生成器および訂正検証器
20 ライン

* 21 ライン

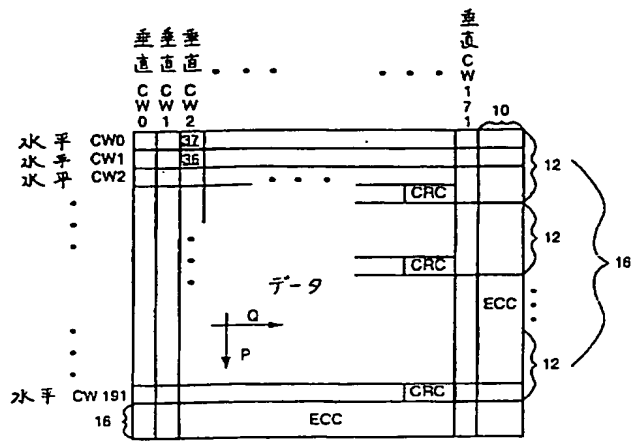
【図 1】



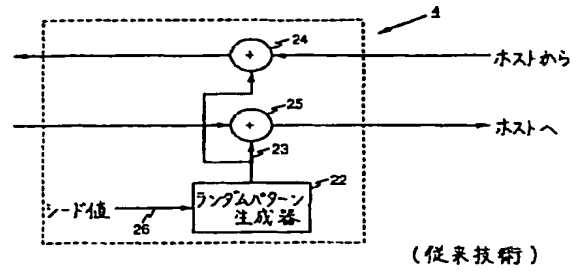
【図 2】



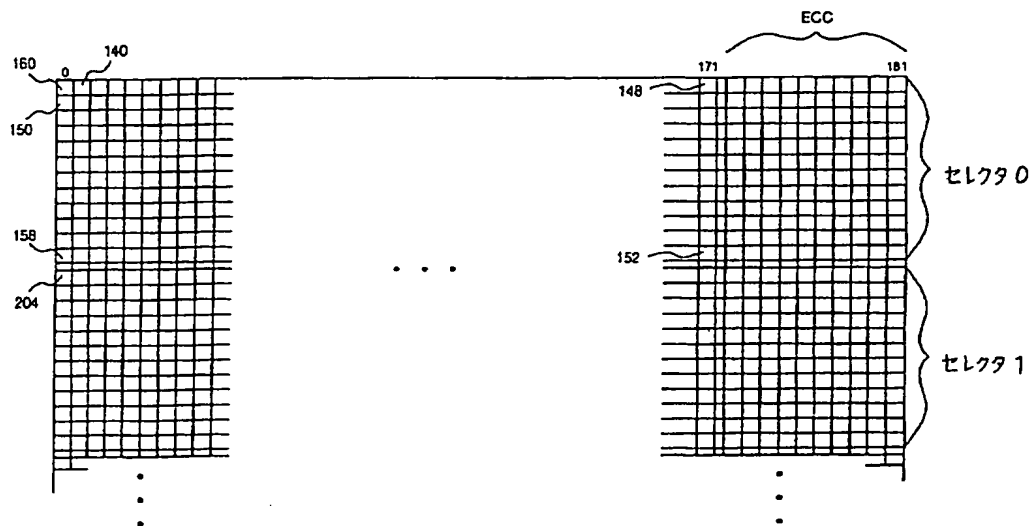
【図 3 A】



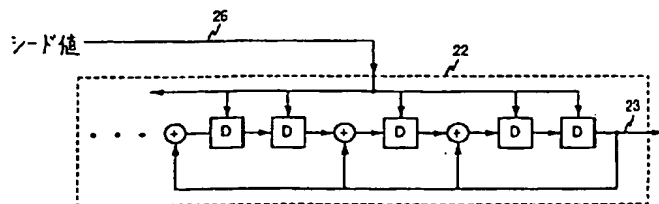
【図 3 C】



【図 3 B】

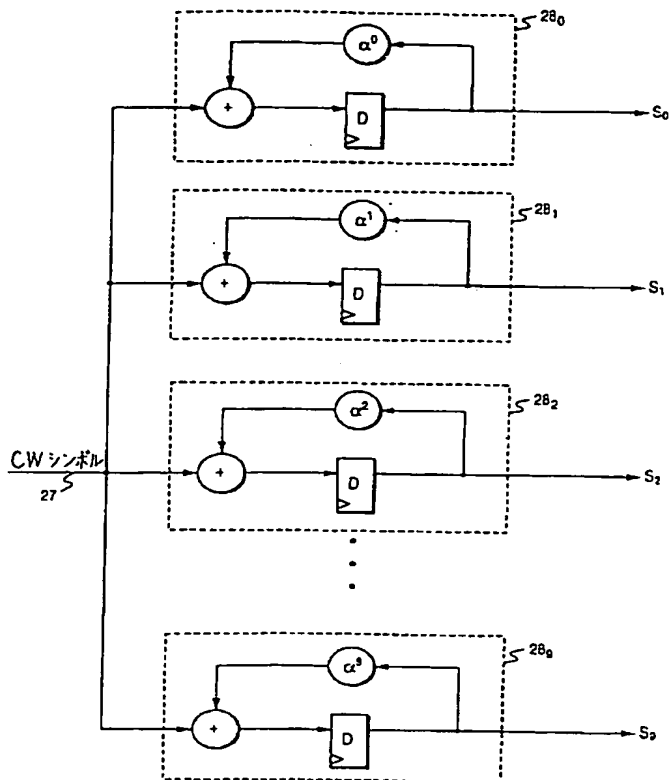


【図 3 D】



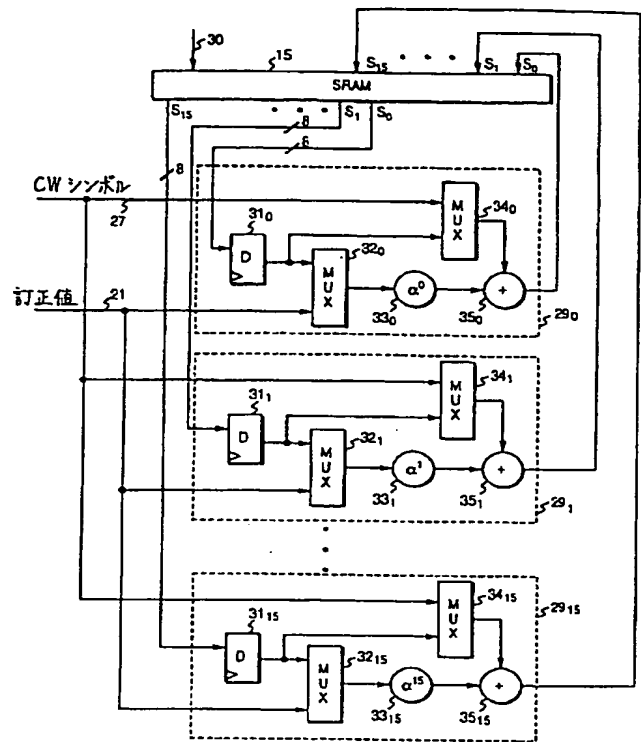
(従来技術)

【図 4】

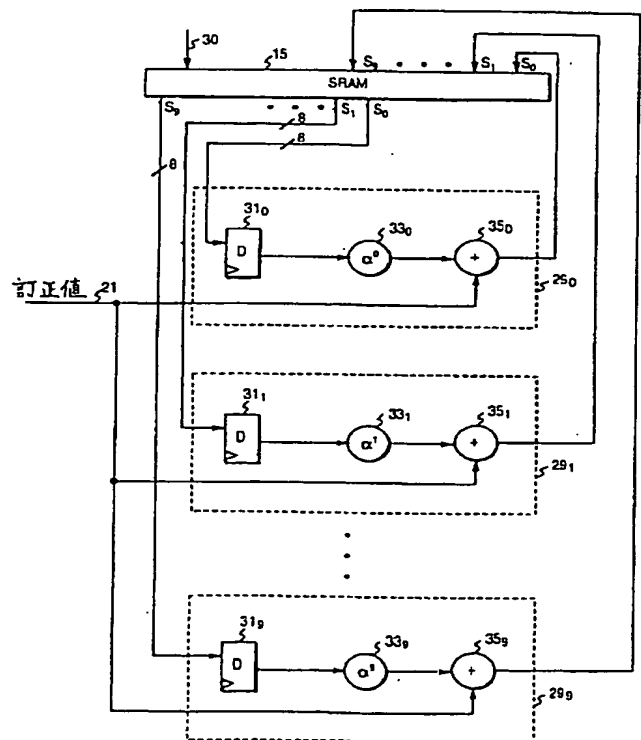


(従来技術)

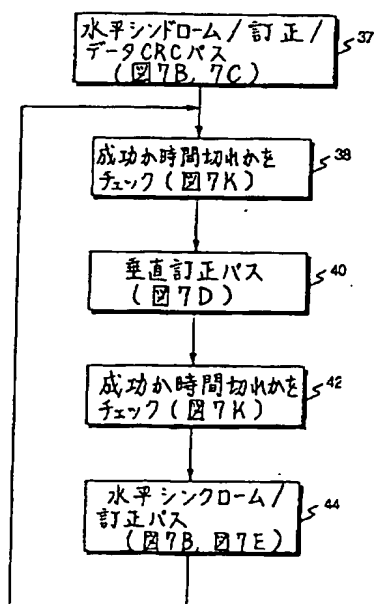
【図 5】



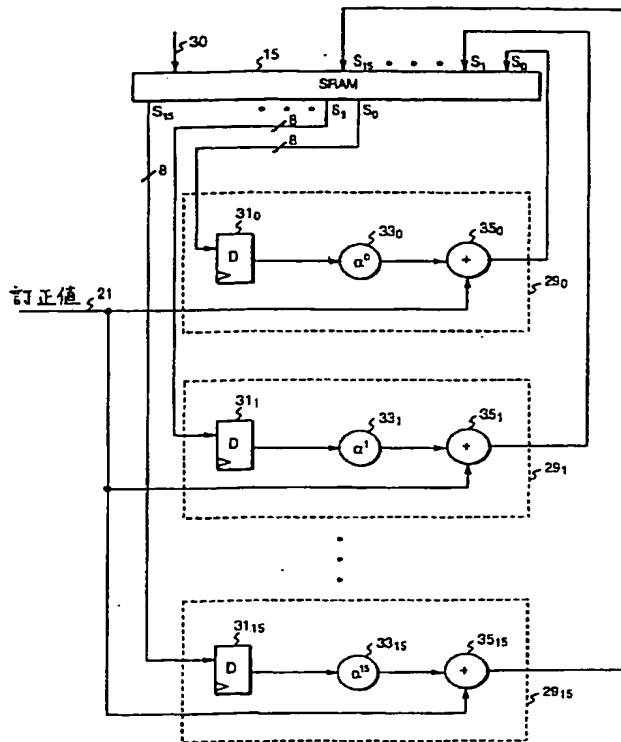
【図 6 A】



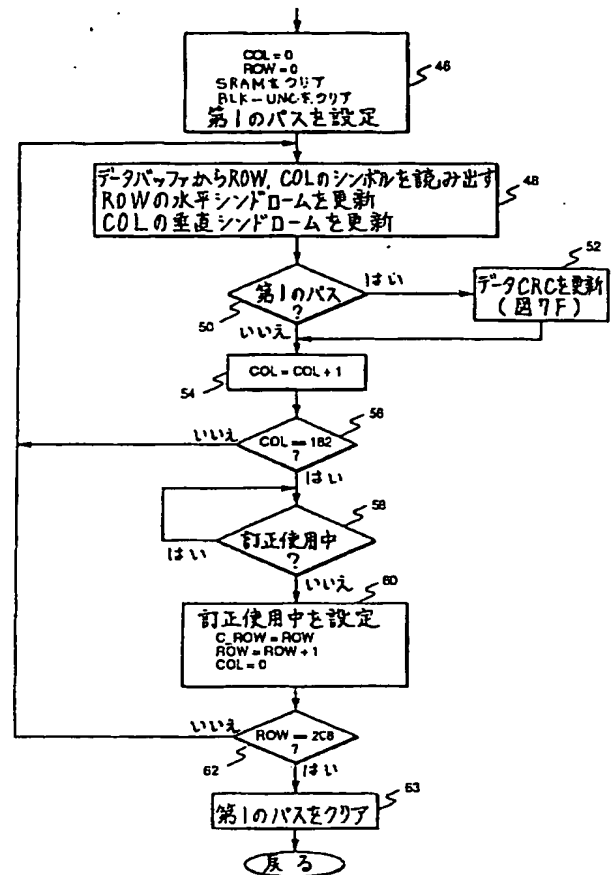
【図 7 A】



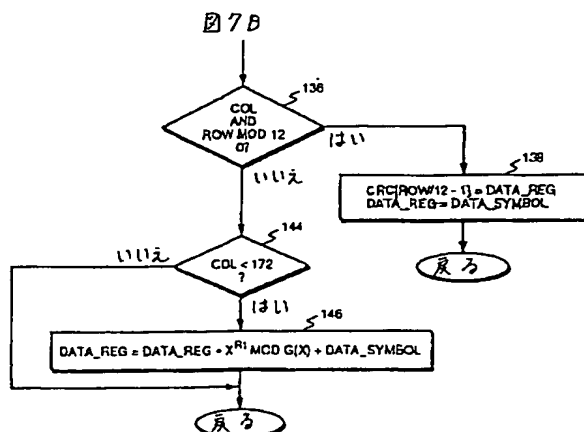
【図6B】



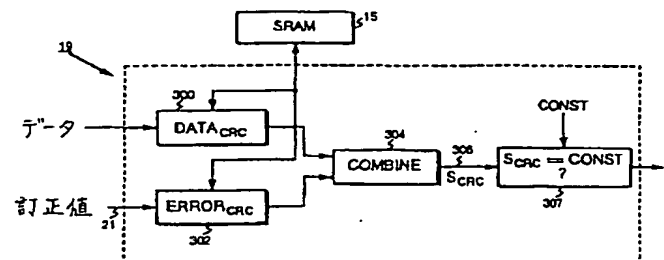
【図7B】



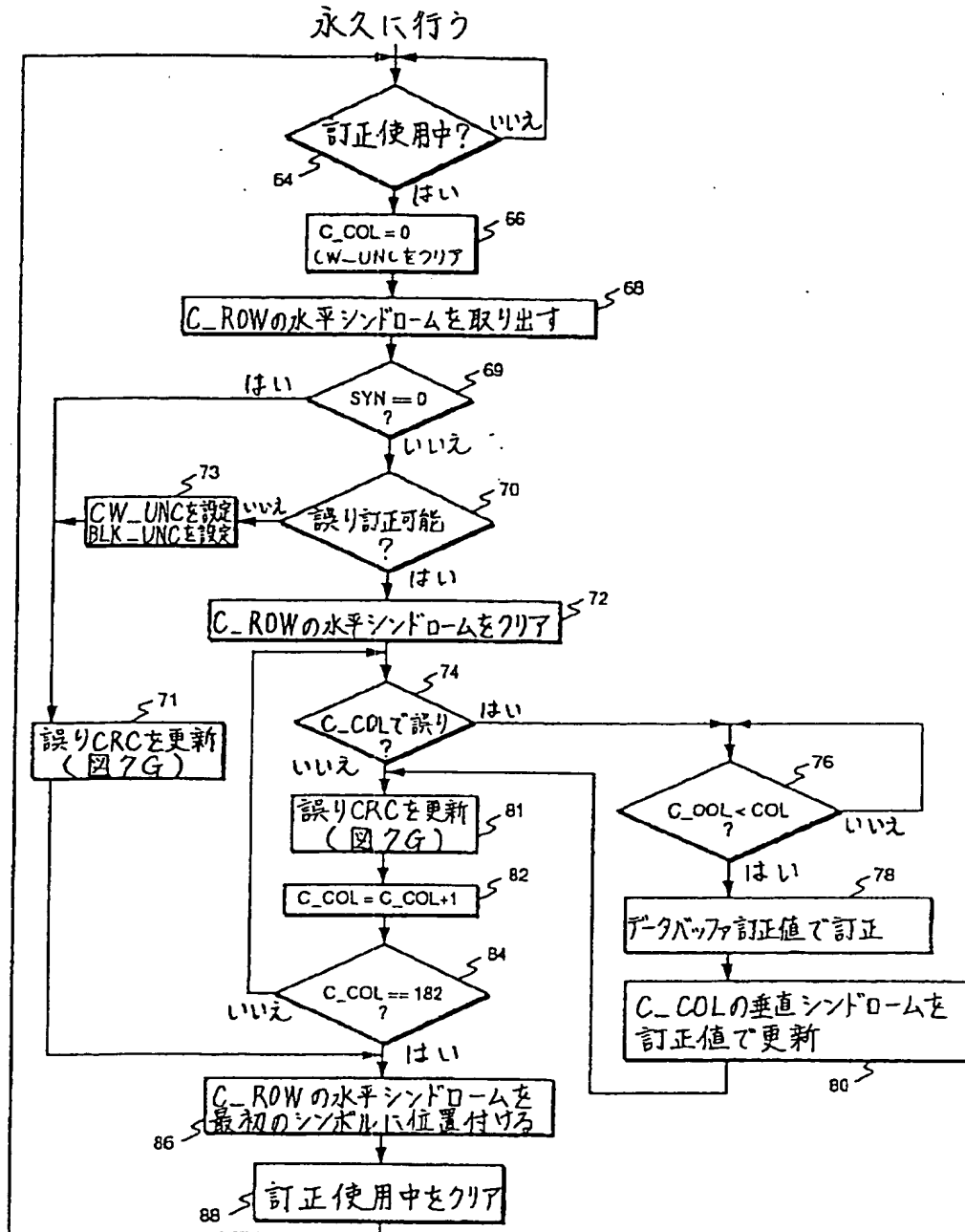
【図7F】



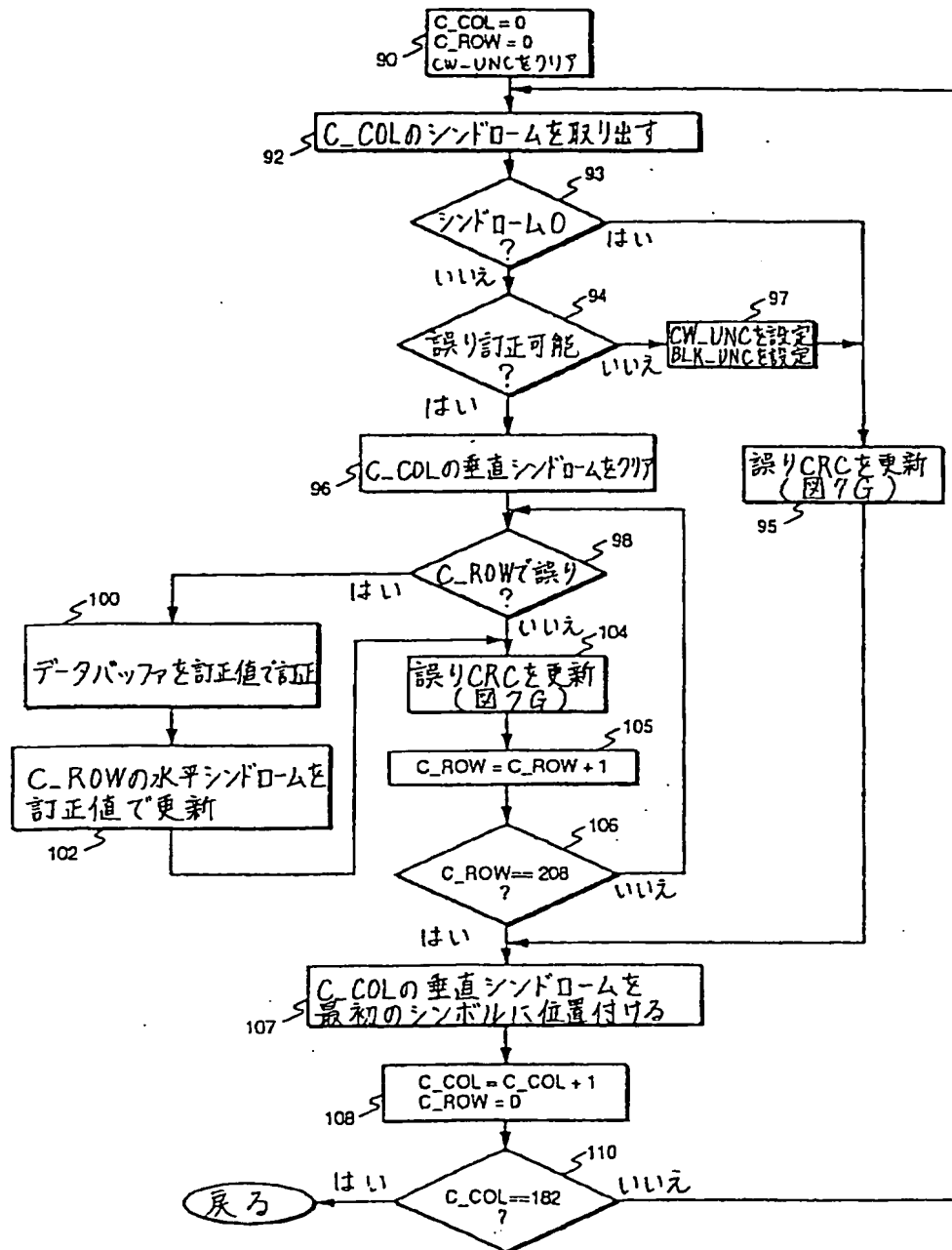
【図9】



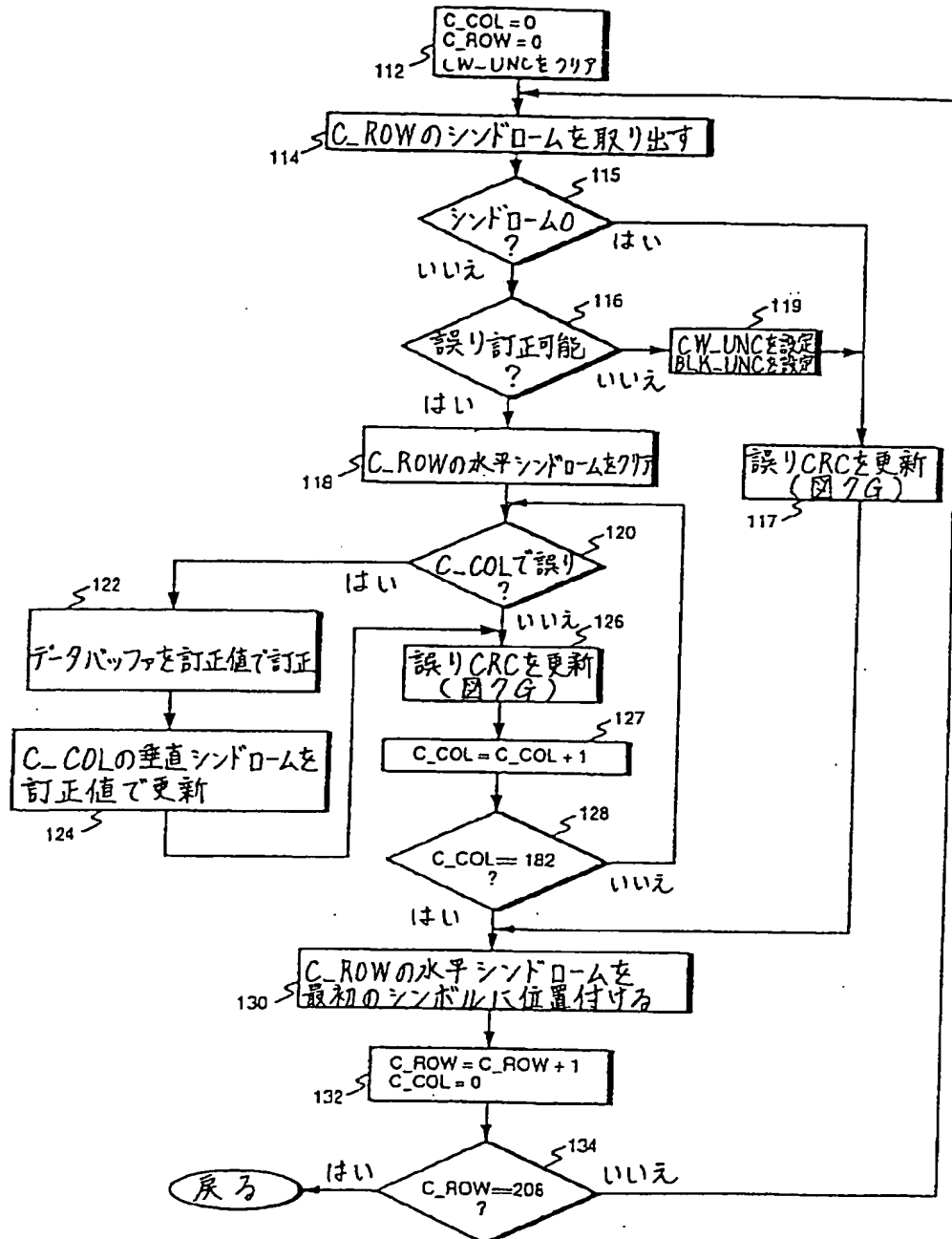
【図7C】



【図7D】



【図7E】



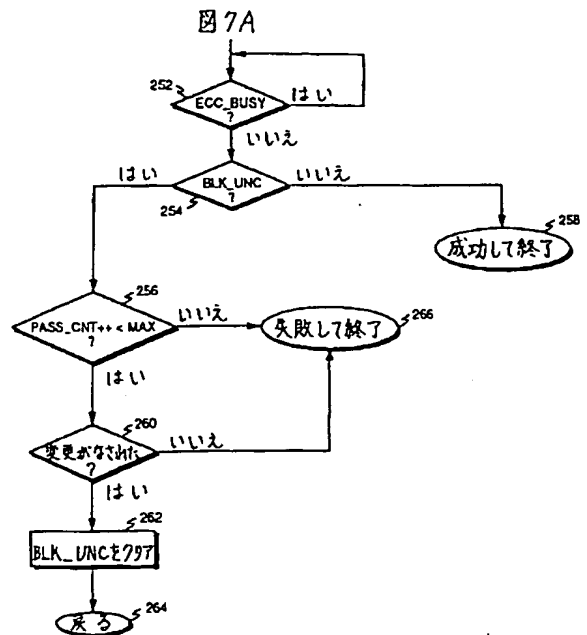
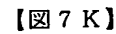
【図 7 H】



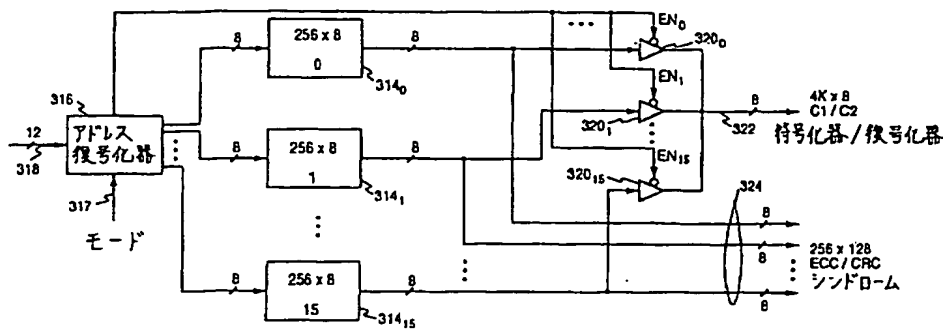
```

graph TD
    Start([スタート]) --> C_COL_171{C.COL = 171?}
    C_COL_171 -- はい --> X11D1_MOD_GQ[X11D1 MOD G(Q)]
    C_COL_171 -- いいえ --> X_R1_MOD_GQ[XR1 MOD G(Q)]
    X11D1_MOD_GQ --> DATA_REG_CRC_ROW_12[DATA_REG = CRC[C_ROW/12]  
S_CRC = DATA_REG + ERR_REG]
    DATA_REG_CRC_ROW_12 --> CRC_ERR_CRC_BLK_UNC_CRC[CRC_ERR CRC  
BLK_UNC_CRC 設定]
    CRC_ERR_CRC_BLK_UNC_CRC --> X_LP12_R1_MOD_GQ[XLP12_R1 MOD G(Q)]
    X_LP12_R1_MOD_GQ --> CRC_CRC_ROW_12_ERR_REG[CRC[C_ROW/12] = ERR_REG]
    X_R1_MOD_GQ --> CRC_CRC_ROW_12_ERR_REG
    CRC_CRC_ROW_12_ERR_REG --> C_ROW_PLUS_12[C_ROW = C_ROW + 12]
    C_ROW_PLUS_12 --> C_ROW_GT_191{C_ROW > 191?}
    C_ROW_GT_191 -- はい --> End([戻る])
    C_ROW_GT_191 -- いいえ --> ERR_REG_CRC_CRC_ROW_12[ERR_REG = CRC[C_ROW/12]]
    ERR_REG_CRC_CRC_ROW_12 --> C_COL_171

```



【図 12A】



【図 13 A】

```

edc_err.vhd                                Wed Jul 16 10:32:32 1997                                1
-- VHDL Modul Created from SDC Symbol edc_err.symb -- Jan 20 09:14:19 1997

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

entity EDC_ERR is
    Port (
        CLK      : In      std_logic;
        CTL      : In      std_logic_vector (5 downto 3);
        D10      : In      std_logic_vector (7 downto 0);
        D11      : In      std_logic_vector (7 downto 0);
        DVD      : In      std_logic;
        RST      : In      std_logic;
        EP_1M    : In      std_logic_vector (3) downto 0;
        DO       : Out     std_logic_vector (3) downto 0);
end EDC_ERR;

architecture BEHAVIORAL of EDC_ERR is

    type array_16x16 is array (0 to 15) of std_logic_vector(15 downto 0);
    type array_32x12 is array (0 to 31) of std_logic_vector(11 downto 0);

    constant r2_1_tbl : array_16x16 :=
        "1000000000000101",
        "1000000000011111",
        "1000000000011011",
        "1000000000111001",
        "1000000001100011",
        "1000000100000011",
        "1000001000000011",
        "1000010000000011",
        "1001000000000011",
        "1001000000000011",
        "1010000000000011",
        "1100000000000011",
        "0100000000000011",
        "0000000000000011",
        "0000000000000011";

    constant d1_1_tbl : array_16x16 :=
        "011010101010011",
        "110101010100101",
        "010101010100001",
        "101010101001001",
        "110101010100001",
        "001010101001011",
        "010101010001110",
        "101010100011100",
        "110101000111101",
        "001010001111111",
        "010101001111110",
        "101001111111101",
        "110011111111101",
        "000011111111011",
        "000111111110110";

    constant d1r2_1_tbl : array_16x16 :=
        "001111111101100";

    constant d1r2_2_tbl : array_16x16 :=
        "011111111101100";

    constant d1r2_3_tbl : array_16x16 :=
        "001111111101100";

    constant d1r2_4_tbl : array_16x16 :=
        "001111111101100";

    constant d1r2_5_tbl : array_16x16 :=
        "001111111101100";

    constant d1r2_6_tbl : array_16x16 :=
        "001111111101100";

    constant d1r2_7_tbl : array_16x16 :=
        "001111111101100";

    constant d1r2_8_tbl : array_16x16 :=
        "001111111101100";

    constant d1r2_9_tbl : array_16x16 :=
        "001111111101100";

    constant d1r2_10_tbl : array_16x16 :=
        "001111111101100";

    constant d1r2_11_tbl : array_16x16 :=
        "001111111101100";

    constant d1r2_12_tbl : array_16x16 :=
        "001111111101100";

    constant d1r2_13_tbl : array_16x16 :=
        "001111111101100";

    constant d1r2_14_tbl : array_16x16 :=
        "001111111101100";

    constant d1r2_15_tbl : array_16x16 :=
        "001111111101100";

    constant d1r2_16_tbl : array_16x16 :=
        "001111111101100";

    constant d1r2_17_tbl : array_16x16 :=
        "001111111101100";

    constant d1r2_18_tbl : array_16x16 :=
        "001111111101100";

    constant d1r2_19_tbl : array_16x16 :=
        "001111111101100";

    constant d1r2_20_tbl : array_16x16 :=
        "001111111101100";

    constant d1r2_21_tbl : array_16x16 :=
        "001111111101100";

    constant d1r2_22_tbl : array_16x16 :=
        "001111111101100";

    constant d1r2_23_tbl : array_16x16 :=
        "001111111101100";

    constant d1r2_24_tbl : array_16x16 :=
        "001111111101100";

    constant d1r2_25_tbl : array_16x16 :=
        "001111111101100";

    constant d1r2_26_tbl : array_16x16 :=
        "001111111101100";

    constant d1r2_27_tbl : array_16x16 :=
        "001111111101100";

    constant d1r2_28_tbl : array_16x16 :=
        "001111111101100";

    constant d1r2_29_tbl : array_16x16 :=
        "001111111101100";

    constant d1r2_30_tbl : array_16x16 :=
        "001111111101100";

    constant d1r2_31_tbl : array_16x16 :=
        "001111111101100";

    constant d1r2_32_tbl : array_16x16 :=
        "001111111101100";

    constant d1r2_33_tbl : array_16x16 :=
        "001111111101100";

    constant d1r2_34_tbl : array_16x16 :=
        "001111111101100";

    constant d1r2_35_tbl : array_16x16 :=
        "001111111101100";

    constant d1r2_36_tbl : array_16x16 :=
        "001111111101100";

    constant d1r2_37_tbl : array_16x16 :=
        "001111111101100";

    constant d1r2_38_tbl : array_16x16 :=
        "001111111101100";

    constant d1r2_39_tbl : array_16x16 :=
        "001111111101100";

    constant d1r2_40_tbl : array_16x16 :=
        "001111111101100";

    constant d1r2_41_tbl : array_16x16 :=
        "001111111101100";

    constant d1r2_42_tbl : array_16x16 :=
        "001111111101100";

    constant d1r2_43_tbl : array_16x16 :=
        "001111111101100";

    constant d1r2_44_tbl : array_16x16 :=
        "001111111101100";

    constant d1r2_45_tbl : array_16x16 :=
        "001111111101100";

    constant d1r2_46_tbl : array_16x16 :=
        "001111111101100";

    constant d1r2_47_tbl : array_16x16 :=
        "001111111101100";

    constant d1r2_48_tbl : array_16x16 :=
        "001111111101100";

    constant d1r2_49_tbl : array_16x16 :=
        "001111111101100";

    constant d1r2_50_tbl : array_16x16 :=
        "001111111101100";

    constant d1r2_51_tbl : array_16x16 :=
        "001111111101100";

    constant d1r2_52_tbl : array_16x16 :=
        "001111111101100";

    constant d1r2_53_tbl : array_16x16 :=
        "001111111101100";

    constant d1r2_54_tbl : array_16x16 :=
        "001111111101100";

    constant d1r2_55_tbl : array_16x16 :=
        "001111111101100";

    constant d1r2_56_tbl : array_16x16 :=
        "001111111101100";

    constant d1r2_57_tbl : array_16x16 :=
        "001111111101100";

    constant d1r2_58_tbl : array_16x16 :=
        "001111111101100";

    constant d1r2_59_tbl : array_16x16 :=
        "001111111101100";

    constant d1r2_60_tbl : array_16x16 :=
        "001111111101100";

    constant d1r2_61_tbl : array_16x16 :=
        "001111111101100";

    constant d1r2_62_tbl : array_16x16 :=
        "001111111101100";

    constant d1r2_63_tbl : array_16x16 :=
        "001111111101100";

    constant d1r2_64_tbl : array_16x16 :=
        "001111111101100";

    constant d1r2_65_tbl : array_16x16 :=
        "001111111101100";

    constant d1r2_66_tbl : array_16x16 :=
        "001111111101100";

    constant d1r2_67_tbl : array_16x16 :=
        "001111111101100";

    constant d1r2_68_tbl : array_16x16 :=
        "001111111101100";

    constant d1r2_69_tbl : array_16x16 :=
        "001111111101100";

    constant d1r2_70_tbl : array_16x16 :=
        "001111111101100";

    constant d1r2_71_tbl : array_16x16 :=
        "001111111101100";

    constant d1r2_72_tbl : array_16x16 :=
        "001111111101100";

    constant d1r2_73_tbl : array_16x16 :=
        "001111111101100";

    constant d1r2_74_tbl : array_16x16 :=
        "001111111101100";

    constant d1r2_75_tbl : array_16x16 :=
        "001111111101100";

    constant d1r2_76_tbl : array_16x16 :=
        "001111111101100";

    constant d1r2_77_tbl : array_16x16 :=
        "001111111101100";

    constant d1r2_78_tbl : array_16x16 :=
        "001111111101100";

    constant d1r2_79_tbl : array_16x16 :=
        "001111111101100";

    constant d1r2_80_tbl : array_16x16
```

[illegible]

【図 1 3 D】

```

edc_err.vhd      Wed Jul 16 10:32:32 1997      4

-- NMT CW:      U25A2 (0..1,1) (up 16 rows, right 2 cols)
-- NMT PASS:     U25A2 (0..0,0)
-- If bypassing col, NMT CW: 32, NMT PASS: 32
-- 0:
-- STEP D1:      D1A2
-- STEP UP:      U25A2
-- NMT CW:      U25A2.D9 (0..0,0)
-- r=430:        D1A2.D9 (0..0,0)
-- NMT PASS:     U25A2.D9 (0..0,0)
-- If bypassing diag, NMT CW: D1
--
-- K2:           alpha*(2^8-16)
-- D1:           alpha*(43^2*8-188)
-- D1R2:         alpha*(43^2*8-1*8-164)
-- D9:           alpha*(43^2*8^2-6191)
-- U25R2:        alpha*(2^15-1 - 43^2*8*25*2*8-15583)
-- U26R2:        alpha*(2^15-1 - 43^2*8*26*2*8-14895)
--
-- CTL: 0-NOP, 1-R2, 2-D1, 3-D1R2, 4-U25R2, 5-U26R2, 6-D9, 7-EAR
--
-- DVD:
-- dvd poly: x^33 + x^31 + x^4 + 1
-- NUKR2:
-- STEP:         R1
-- VERT:         D1
-- 1st BYTE:     U12R1
--
-- A1:          alpha*(2^8+1)
-- D1:          alpha*(1^172*8+1376)
-- U12R1:        alpha*(2^11-1 - 12^172*8+2^8-2147467143)
--
-- CTL: 0-NOP, 1-LOAD, 2-R1, 3-D1, 4-U12R1, 5-EAR, 6-7-NOP
-----
process (r1)
variable sumr2 : std_logic_vector(15 downto 0);
variable sumd1 : std_logic_vector(15 downto 0);
variable sumd1r2 : std_logic_vector(15 downto 0);
variable sumd9 : std_logic_vector(15 downto 0);
variable sumu25r2 : std_logic_vector(15 downto 0);
variable sumu26r2 : std_logic_vector(15 downto 0);
begin
sumr2 := "0000000000000000";
sumd1 := "0000000000000000";
sumd1r2 := "0000000000000000";
sumd9 := "0000000000000000";
sumu25r2 := "0000000000000000";
sumu26r2 := "0000000000000000";
for i in 0 to 15 loop
if (r1(i)='1') then
sumr2 := sumr2 xor r2_1_tbl(i);
sumd1 := sumd1 xor d1_1_tbl(i);
sumd1r2 := sumd1r2 xor d1r2_1_tbl(i);
sumd9 := sumd9 xor d9_1_tbl(i);
sumu25r2 := sumu25r2 xor u25r2_1_tbl(i);
sumu26r2 := sumu26r2 xor u26r2_1_tbl(i);
end if;
end loop;
r2_1 <= sumr2;
d1_1 <= sumd1;
d1r2_1 <= sumd1r2;
d9_1 <= sumd9;
end process;

process (r2)
variable sumr2 : std_logic_vector(15 downto 0);
variable sumd1 : std_logic_vector(15 downto 0);
variable sumd1r2 : std_logic_vector(15 downto 0);
variable sumd9 : std_logic_vector(15 downto 0);
variable sumu25r2 : std_logic_vector(15 downto 0);
variable sumu26r2 : std_logic_vector(15 downto 0);
begin
sumr2 := "0000000000000000";
sumd1 := "0000000000000000";
sumd1r2 := "0000000000000000";
sumd9 := "0000000000000000";
sumu25r2 := "0000000000000000";
sumu26r2 := "0000000000000000";
for i in 0 to 15 loop
if (r2(i)='1') then
sumr2 := sumr2 xor r2_2_tbl(i);
sumd1 := sumd1 xor d1_2_tbl(i);
sumd1r2 := sumd1r2 xor d1r2_2_tbl(i);
sumd9 := sumd9 xor d9_2_tbl(i);
sumu25r2 := sumu25r2 xor u25r2_2_tbl(i);
sumu26r2 := sumu26r2 xor u26r2_2_tbl(i);
end if;
end loop;
r2_2 <= sumr2;
d1_2 <= sumd1;
d1r2_2 <= sumd1r2;
d9_2 <= sumd9;
u25r2_2 <= sumu25r2;
u26r2_2 <= sumu26r2;
end process;

process (r1,r2)
variable sumr1 : std_logic_vector(31 downto 0);
variable sumd1 : std_logic_vector(31 downto 0);
variable sumu12r1 : std_logic_vector(31 downto 0);
begin
sumr1 := "00000000000000000000000000000000";
sumd1 := "00000000000000000000000000000000";
sumu12r1 := "00000000000000000000000000000000";
for i in 0 to 15 loop
if (r1(i)='1') then
sumr1 := sumr1 xor r1_dvd_tbl(i);
sumd1 := sumd1 xor d1_dvd_tbl(i);
sumu12r1 := sumu12r1 xor u12r1_dvd_tbl(i);
end if;
if (r2(i)='1') then
sumr1 := sumr1 xor r1_dvd_tbl(i+16);
sumd1 := sumd1 xor d1_dvd_tbl(i+16);
sumu12r1 := sumu12r1 xor u12r1_dvd_tbl(i+16);
end if;
end loop;
r1_dvd <= sumr1;
d1_dvd <= sumd1;
u12r1_dvd <= sumu12r1;
end process;

process
begin
wait until (CLK'event) and (CLK='1');

```

【図 1 3 E】

```

edc_err.vhd      Wed Jul 16 10:32:32 1997      5

if (rst='1') then
  r1 <= "0000000000000000";
  r2 <= "0000000000000000";
elsif (DVD='0') then      -- CD
  case CTL is
    when "00" =>
      r1 <= r2_1;
      r2 <= r2_2;
    when "010" =>
      r1 <= d1_1;
      r2 <= d1_2;
    when "011" =>
      r1 <= d1r2_1;
      r2 <= d1r2_2;
    when "100" =>
      r1 <= u2sr2_1;
      r2 <= u2sr2_2;
    when "101" =>
      r1 <= u2sr2_1;
      r2 <= u2sr2_2;
    when "110" =>
      r1 <= d3_1;
      r2 <= d3_2;
    when "111" =>
      r1 <= r1 xor
        D10(0) & D10(1) & D10(2) & D10(3) & D10(4) & D10(5) & D10(6) & D10(7) &
        D11(0) & D11(1) & D11(2) & D11(3) & D11(4) & D11(5) & D11(6) & D11(7);
      r2 <= r2 xor
        D10(0) & D10(1) & D10(2) & D10(3) & D10(4) & D10(5) & D10(6) & D10(7) &
        D11(0) & D11(1) & D11(2) & D11(3) & D11(4) & D11(5) & D11(6) & D11(7);
    when others =>
      r1 <= r1;
      r2 <= r2;
  end case;
else      -- DVD
  case CTL is
    when "00" =>
      r1 <= SR_1H(15 downto 0);
      r2 <= SR_1H(31 downto 16);
    when "010" =>
      r1 <= r1_dvd(15 downto 0);
      r2 <= r1_dvd(31 downto 16);
    when "011" =>
      r1 <= d1_dvd(15 downto 0);
      r2 <= d1_dvd(31 downto 16);
    when "100" =>
      r1 <= u1r1_dvd(15 downto 0);
      r2 <= u1r1_dvd(31 downto 16);
    when "101" =>
      r1 <= r1 xor "00000000" & D10;
    when others =>
      r1 <= r1;
      r2 <= r2;
  end case;
end if;
end process;

DO <= r2 & r1;

end BEHAVIORAL;

```

フロントページの続き

(71)出願人 595158337

3100 West Warren Aven
ue, Fremont, Californ
ia 94538, U. S. A.

**This Page is Inserted by IFW Indexing and Scanning
Operations and is not part of the Official Record**

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images include but are not limited to the items checked:

- ☐ **BLACK BORDERS**
- ☐ **IMAGE CUT OFF AT TOP, BOTTOM OR SIDES**
- ☐ **FADED TEXT OR DRAWING**
- ☐ **BLURRED OR ILLEGIBLE TEXT OR DRAWING**
- ☐ **SKEWED/SLANTED IMAGES**
- ☐ **COLOR OR BLACK AND WHITE PHOTOGRAPHS**
- ☐ **GRAY SCALE DOCUMENTS**
- ☐ **LINES OR MARKS ON ORIGINAL DOCUMENT**
- ☐ **REFERENCE(S) OR EXHIBIT(S) SUBMITTED ARE POOR QUALITY**
- ☐ **OTHER: _____**

IMAGES ARE BEST AVAILABLE COPY.

As rescanning these documents will not correct the image problems checked, please do not report these problems to the IFW Image Problem Mailbox.